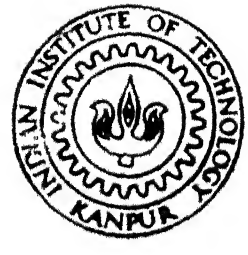
ON SUBMODULAR, BISUBMODULAR SYSTEMS AND SUBMODULAR, NETWORK FLOWS

by

S. RAJESH



DEPARTMENT OF MATHEMATICS

INDIAN INSTITUTE OF TECHNOLOGY KANPUR

JULY, 1997

ON SUBMODULAR, BISUBMODULAR SYSTEMS AND SUBMODULAR, NETWORK FLOWS

A Thesis Submitted

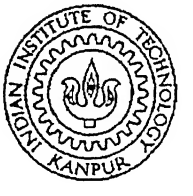
in Partial Fulfilment of the Requirements

for the Degree of

DOCTOR OF PHILOSOPHY

by

S. RAJESH



to the

DEPARTMENT OF MATHEMATICS
INDIAN INSTITUTE OF TECHNOLOGY KANPUR

July, 1997

CERTIFICATE

It is certified that the work contained in the thesis entitled "*On Submodular, Bisubmodular Systems and Submodular, Network Flows*", by *S. Rajesh*, has been carried out under my supervision and that this work has not been submitted elsewhere for a degree.

Prabha Sharma

Dr. PRABHA SHARMA
Professor
Department of Mathematics
I.I.T. Kanpur

July, 1997

SYNOPSIS

In this dissertation an attempt has been made to study the properties of the submodular functions having non zero values at the empty set. When the value of f at the empty set is nonnegative the greedy algorithm will yield an optimal solution to the Linear Programming Problem (LPP) associated with the polyhedron $P(f)$. But the greedy algorithm may fail to solve the LPP when the submodular function has negative value at the empty set. We investigate the reasons why this happens. An algorithm to solve this class of LPP using minimisation of a submodular function as an oracle is given. Some example where the submodular functions with negative values at the empty set occur are given We show that the system

$$\{Max\ cx \mid x \in P(f),\ f(\phi) < 0\}$$

is also totally dual integral. We give connectedness properties for the submodular system (f, D) when, $f(\phi) < 0$, and show that the base polyhedron $B(f)$ corresponding to the polyhedron $P(f)$ may be empty even when $P(f) \neq \phi$.

Following this, properties of bisubmodular functions with negative values at the empty set have been studied. The relationship between greedy algorithm and bisubmodularity was first pointed out by Kabadi and Chandresekeran. They have shown that when $f(\phi, \phi) = 0$, then the greedy algorithm implies bisubmodularity and vice versa. In this chapter an attempt has been made to study the relationship between bisubmodularity and the greedy algorithm when $f(\phi, \phi) \neq 0$.

The LPP associated with bisubmodular polyhedron can be solved by generalised greedy algorithm when the value of f at empty set is nonnegative. But when $f(\phi, \phi) < 0$ the generalised greedy algorithm may fail to produce even a feasible solution. We have shown that it is not easy to solve the LPP associated with such functions as it is in the case for

submodular functions. However, we give efficient algorithm for solving the LPP

$$\{Max\ cx \mid x \in \mathcal{R}^n,\ x(A,B) \leq f(A,B),\ (A,B) \subseteq (X,Y)\}$$

Where $X = \{e_i \mid c_{e_i} \geq 0\}$, $Y = \{e_i \mid c_{e_i} < 0\}$ and $f(\phi,\phi) < 0$.
Some of the existing results for bisubmodular functions assumed to have zero value at the empty set is extended to functions having nonzero values at the empty set.

We next study the Bisubmodular Systems. First we prove some properties of increasing bisubmodular functions and their associated polyhedra. We have shown that many properties of the polyhedron associated with increasing bisubmodular function and the polymatroid polyhedron are similar. It has been proved that the tight sets corresponding to the non zero components of $v \in P(f)$ are non-canceling. This property helps us to define support function, closoure function and the partial order for the extreme points corresponding to the increasing bisubmodular polyhedron.
Min-Max theorem for the bisubmodular system is proved in a way different from the existing proofs. Relationship between separation problem and bisubmodular function minimization is given. At the end we minimise bisubmodular functions with respect to some simple constraints that arise in practical problems.

In the last chapter we develop a variant of cycle-canceling algorithm, for solving minimum cost network and submodular flow problems. Our cycle-canceling algorithm for the minimum cost network flow problem performs the following steps on the residual network. We look for augumenting cycles as well as dynamically update potentials of the nodes blocking the search. The basic idea is to reduce the optimality violation of all arcs incident to the node blocking the search by half. The main feauture of our algorithm is that it simultaneously finds negative cycles and tightens optimality violations by using the basic

graph search algorithm. This algorithm has also been extended to solve the minimum cost submodular flow problems. Like all the algorithms for the minimum cost submodular flow problem this algorithm also requires an oracle for minimising a submodular function. Since computational experimentation has been the primary method for comparative evolution of network flow algorithms, a study on computational behaviour of our algorithm has been carried out. We compare our algorithm with the relaxation algorithm for solving minimum cost network flow problem. The test problems used are the standard bench marks generated by the publically available Netgen program.

At the end we give concluding remarks and highlight scope for future research.

DEDICATED TO
MY PARENTS

ACKNOWLEDGEMENTS

I have great pleasure in expressing my sincere gratitude to my thesis supervisor Prof.(Mrs.) **Prabha Sharma**, for excellent guidance, constructive criticism and constant encouragement throughout the course of this study. She has always given me free hand to develop my own ideas and has always helped clarify my concepts.

I take this opportunity to thank Dr S.K Gupta, Dr Ravindra K. Ahuja and Dr P. Shanmugaraj for their suggestions and help in my thesis work. My sincere thanks are due to Drs. V.Raghavendra, S.Madan, D.kundu, Santokh Singh, B.V Rathish Kumar and other members of faculty who have helped and encouraged me in many ways.

I express my sincere thanks to my senior Dr P.T Sokkalingam for the help I received from him in my thesis . It is a pleasure to acknowledge my colleagues Guha, Sokks, Durai, Sivaguru, Arumugam, Kuppi, Murthy and many others for their help and company during my stay in the campus.

I thank all non teaching staff of my department for their help I also thank canteen Narayanan for his valuable service.

Many friends have made my stay in IIT Kanpur a memorable one. It will be very difficult to forget friends like "A.P", Ravi, Umamahesh, Raghu, Captain Ramesh, Guru, Kasi, Seeni, Ilango, Ponds, Kadi Siva, Durai, Rsel, Manoravi, Sokks, Tamil, Sampath, "Periyavar" Elango, Immi(shivde), Yogi, Arumugam, Kuppi, "Koys", Justin, Vivek. Dhans, Anbu "Bucket", Punns, Kottes, Kumar, Shankar, Johnson, Sivaguru, Periyaps, Kannan, Jeebu, Yettu, Suresh, Subash, Appaji, Murty, Nattu, Jeer, Venkatesh(NSI), Ramesh Rao, Ravichandran, Nari, Siva, Balaji, Murali, Marthandam, Muthu, G.K, Deepak, Joyadeep and many others who do not figure in this list.

I also thanks Mrs Ramesh, Mrs Usha Elango, Mrs Anitha venkatesh and Mrs Sridevi Punniyamurthy for their hospitality.

I am grateful to all my school and college teachers who have encouraged me throughout

my studies.

I am grateful to my father-in-law and mother-in-law for their constant support and encouragement.

I would like to acknowledge my parents for their affection and encouragement and for instilling in me a love for learning.

Finally I offer my heartfelt thanks to my wife for her love and understanding nature during my research.

S Rajesh

Contents

Chapter 1

INTRODUCTION

1.1 Literature survey

Edmonds in his pioneering paper in 1970 introduced submodular functions and the submodular polyhedron. In particular the matroid polyhedron is a submodular polyhedron and the matroid rank function, a submodular function on the subsets of the ground set E of the matroid. He showed that a submodular polyhedron has the total dual integrality property and in case the submodular function defining the polyhedron is integer valued then every extreme point of the polyhedron is an integer valued vector. Edmonds also constructed the greedy algorithm to solve the LPP associated with a submodular polyhedron. Further research focused on finding the larger class of polyhedra which have the total dual integrality property and also the greedy property in a generalised sense. Some of the results are in [37], [49],[50], [53] and [46].

Submodular functions have a lot of applications in practical engineering problems. See [35] [43] and [48] for details. The importance of submodular functions has been widely recognized in recent years in combinatorial optimization and other fields of combinatorial analysis.

Another aspect of submodularity theory is minimization of a submodular function. In [32] Grötschel et.el have devised a strongly polynomial algorithm for minimising a submodular

function, which relies heavily on the ellipsoidal algorithm. Oracle for minimising a submodular function is required for solving submodular flow problems and many other related problems.

The membership problem for a submodular polyhedra is, given a vector $x \in \mathcal{R}^n$, determine whether $x \in P(f)$ or not. It has been shown that the membership problem is equivalent to minimising a submodular function [42].

Pseudo-polynomial algorithms for the membership problem are given in [13], [14], [45] and [51], which can also be used for minimising a submodular function.

Another aspect of submodularity theory which has been extensively studied is the polyhedral structure of the submodular polyhedron. For details you may see [54], [25] and [57].

In the direction towards generalizing the results based on submodular functions, comes the theory of bisubmodular functions, and bisubmodular polyhedra.

Welsh [56] considered a general class of polyhedra for which the greedy algorithm works. This class of polyhedra is exactly that of bisubmodular polyhedra determined by bisubmodular functions. The relationship between greedy algorithm and bisubmodularity was first pointed out by Kabadi [36], Chandrasekaren and Kabadi [12]. Various aspects of these bisubmodular polyhedra and the bisubmodular functions have been studied in [17] and [47]. Different approaches to bisubmodular systems were given by L. Qi [46], Fujishige [4] and Guha et.el [33].

Another major area of submodularity theory is submodular flows. Submodular flow problems are classical generalization of network flow problems. Network flow problems with submodular boundary constraints, are called submodular flow problems. It includes the (poly)-matroid intersection problem of J.Edmonds [19] and the submodular flow problem of J. Edmonds and R. Giles [18].

Researchers have proposed many algorithms to solve the minimum cost network flow prob-

lem. Most of the fundamental algorithms such as primal network simplex algorithms, the primal-dual algorithms, the successive shortest-path algorithm, the out of kilter algorithms and cycle-cancelling algorithms were developed in 1950's and 1960's.

The development of the first polynomial algorithm by Edmonds and Karp [38] have created interest in developing polynomial and strongly polynomial versions of traditional algorithms.

For solving the submodular flow problem the algorithms such as primal network simplex algorithm, primal-dual algorithm and cycle-cancelling algorithm for min-cost network flow problems have been extended. These algorithms were developed in 1980's and need an oracle for minimising a submodular function.

The first strongly polynomial algorithm for the submodular flow problem was given by Frank and Tordos [58]. This algorithm relies heavily on the ellipsoid algorithm for minimising a submodular function.

1.2 Preliminaries

In this section we give basic definitions and results from submodular, bisubmodular theory, graph theory, complexity theory and formulations of minimum cost flow problem and its submodular version.

Submodular systems

Let E be a non empty finite set and D be the collection of all subsets of E. D is a distributive lattice with set union and intersection as the lattice operations, that is, for any X,Y $\in$ D we have $X \cup Y$ and $X \cap Y \in D$.

Let $f : D \to \Re$. f is said to be submodular on D if

$\forall X, Y \in D : f(X) + f(Y) \geq f(X \cup Y) + f(X \cap Y)$.

The pair (f, D) is called a submodular system.

Definition 1.2.1. *A family $\mathcal{F} \subseteq D$ is called an intersecting family if for each intersecting $X, Y \in \mathcal{F}$ (that is $X, Y \in \mathcal{F}$ and $X \cap Y \neq \phi$) we have $X \cup Y, X \cap Y \in \mathcal{F}$. A submodular function $f : \mathcal{F} \to \Re$ on the intersecting family f is called an intersecting submodular function.*

A family $\mathcal{F} \subseteq 2^E$ is called an crossing family if for each crossing $X, Y \in \mathcal{F}$ (that is $X, Y \in \mathcal{F}$ and $X \cap Y \neq \phi, X - Y \neq \phi, Y - X \neq \phi$ and $X \cup Y \neq E$) we have $X \cup Y, X \cap Y \in \mathcal{F}$. A submodular function $f : \mathcal{F} \to \Re$ on the crossing family $\mathcal{F}$ is called a crossing submodular function.

Let f be a submodular function defined on D.

Define a polyhedron in $\Re^E$ as:

$$P(f) = \left\{ x \mid x \in \Re^E, \forall X \in D : \sum_{i \in X} x_i \leq f(X) \right\}$$

$P(f)$ is the submodular polyhedron associated with submodular function f.

Let us define the base polyhedron $B(f)$ associated with $P(f)$ as,

$$B(f) = \{x \mid x \in P(f),\ x(E) = f(E)\}$$

Let $v \in P(f)$. A set $B \subseteq E$ is said to be a v-tight set if $v(A) = f(A)$.

$v \in P(f)$ is said to be an extreme point of $P(f)$ if it can not be expressed as a convex combination of the other two points *in $P(f)$.*

Note that the value of f on the empty set need not be zero. Where ever it is so, it will be specifically mentioned in the thesis.

Directed Sets

Definition 1.2.2. *Directed set: Let E be a finite set with cardinality of E=$\mid E \mid = n$. A directed set(or diset) $X = (X_1, X_2)$ of E is an ordered pair of disjoint subset of E, that is*

$X_1 \subseteq E$ and $X_2 \subseteq E$ and $X_1 \cap X_2 = \phi$. We also identify a diset of $X = (X_1, X_2)$ of E with a n-dimensional $(0, \pm 1)$ vector as follows:

$$X(e) = \begin{cases} 1 \text{ if } e \in X_1 \\ -1 \text{ if } e \in X_2 \\ 0 \text{ otherwise} \end{cases}$$

Let D(E) denote the collection of all directed subsets of E. We will also regard the elements of D(E) as collection of all n-dimensional (0 ,$\pm$1) characteristic vectors defined above.

Operations on disets:.

Suppose X=(X_1, X_2) and Y=(Y_1, Y_2) are two disets.

(i) X=-Y if $X_1 = Y_2$ *and* $X_2 = Y_1$

(ii) $X \leq Y$ *if* $X_1 \subseteq Y_2$ *and* $X_2 \supseteq Y_1$

(iii) $X \wedge Y = (X_1 \cap Y_1, X_2 \cup Y_2)$

(iv) $X \vee Y = (X_1 \cup Y_1, X_2 \cap Y_2)$

(v) $X \subseteq Y$ if and only if $X_1 \subseteq Y_1$ *and* $X_2 \subseteq Y_2$

(vi) $X \cap Y = (X_1 \cap Y_1, X_2 \cap Y_2)$

$$X \cap Y(e) = \begin{cases} 1 \text{ if } X(e) = Y(e) = 1 \\ -1 \text{ if } X(e) = Y(e) = -1 \\ 0 \text{ otherwise} \end{cases}$$

(vii) $X \cup Y = (X_1 \cup Y_1 \backslash X_2 \cup Y_2, X_2 \cup Y_2 \backslash X_1 \cup Y_1)$

$$X \cup Y(e) = \begin{cases} 1 \text{ if } X(e) + Y(e) > 0 \\ -1 \text{ if } X(e) + Y(e) < 0 \\ 0 \text{ if } X(e) + Y(e) = 0 \end{cases}$$

(viii) $X\backslash Y = (X_1\backslash(Y_1 \cup Y_2), X_2\backslash(Y_1 \cup Y_2)$

$$X\backslash Y(e) = \begin{cases} 1 \text{ if } X(e) = 1 \text{ and } Y(e) = 0 \\ -1 \text{ if } X(e) = -1 \text{ and } Y(e) = 0 \\ 0 \text{ otherwise} \end{cases}$$

Definition 1.2.3.

(1) Two disets X, Y are said to be intersecting if $X \not\subset Y$, $Y \not\subset X$ and $X \cap Y \neq \phi$.

(2) Two disets $X = (X_1, X_2), Y = (Y_1, Y_2)$ are said to be non cancelling if and only if $X_1 \cap Y_2$ and $X_2 \cap Y_1 = \phi$. Otherwise X, Y are cancelling disets.

(3)$\bar{X}$=absolute(X)=$X_1 \cup X_2$, So

$|X| = |\bar{X}| = |X_1 \cup X_2| = |X_1| + |X_2|$

(4) Let $\mathcal{T}$ denote some collection of disets. $\mathcal{T}$ is said to be an intersecting family if for any two disets $X, Y \in \mathcal{T}$ and $X \cap Y \neq \phi$ implies $X \cup Y$ and $X \cap Y \in \mathcal{T}$.

(5) A diset $X = (X_1, X_2)$ is said to be an orthant if $|X_1 \cup X_2| = |E| = n$.

In what follows we give some properties of disets with out their proofs.

Let X,Y,Z be three disets of D(E)

(i) $X \subseteq Y \subseteq Z \Rightarrow X \subseteq Z$.

(ii) $X \subseteq Y \subseteq X \Rightarrow X = Y$.

(iii) $X \cup Y = Y \cup X$

(iv) $X \cap Y = Y \cap X$

(v) $X \cap Y \subseteq X$ *and* $X \cap Y \subseteq Y$

(vi) $X \cap Y \subseteq X \cap Y$

(vii) $X \cap (Y \cap Z) = (X \cap Y) \cap Z$

(viii) $X \cup Y = \phi$ *if and only if* $X = -Y$

(ix) $(X \cup Y) \cap Z = \{(X \cup Z) \cap (Y \cup Z)\} \cap \{(X \cap Z) \cup (Y \cap Z)\}$

(x) $(X \cap Y) \cup Z = \{(X \cup Z) \cap (Y \cup Z)\} \cup \{(X \cap Z) \cup (Y \cap Z)\}$

The following properties are not true in general.

(i)$X \subseteq X \cup Y$ *or* $Y \subseteq X \cup Y$

(ii)$(X \cup Y) \cup Z = X \cup (Y \cup Z)$

$(iii) X \cup (Y \cap Z) = (X \cup Y) \cap (X \cup Z)$

$(iv) X \cap (Y \cup Z) = (X \cap Y) \cup (X \cap Z)$

Bisubmodular Systems

Definition 1.2.4. *Let $f : D(E) \to \mathcal{R}$ be a biset function on $D(E)$. f is said to be a bisubmodular (or directed submodular) function if for any $X = (X_1, X_2)$ and $Y = (Y_1, Y_2) \in D(E)$*

$$f(X_1X_2) + f(Y_1, Y_2) \geq f(X_1 \cup Y_1 \backslash X_2 \cup Y_2, X_2 \cup Y_2 \backslash X_1 \cup Y_1) + f(X_1 \cap Y_1, X_2 \cap Y_2)$$

The pair $(f, D(E))$ is called a bisubmodular system.

f is said to be non-decreasing (increasing) if $f(X) \leq f(Y)$ $\forall X \subseteq Y$, $X, Y \in D(E)$.

Definition 1.2.5. *Let f be a bisubmodular function on $D(E)$*

Define a polyhedron in $\Re^E$ by

$$P(f) = \left\{ x \mid x \in \Re^E, \forall (X, Y) \in D(E) : \sum_{i \in X} x_i - \sum_{i \in Y} x_i \leq f(X, Y) \right\}$$

$P(f)$ is the bisubmodular polyhedron associated with bisubmodular function f.

Let $v \in P(f)$. A set $(A, B) \in D(E)$ is said to be a v-tight set if $v(A, B) = f(A, B)$.

$v \in P(f)$ is said to be a extreme point of $P(f)$ if it is not a convex combination of two other points *in $P(f)$.*

In what follows is an example of a bisubmodular function.

Let E=(1,2,3). Define f as follows:

$f(i, \phi) = f(\phi, i) = 1, \quad i = 1, 2, 3$

$f(ij, \phi) = f(\phi, ij) = f(i, j) = 2, \quad i, j = 1, 2, 3 \; i \neq j$

$f(ij, k) = f(\phi, ijk) = f(i, jk) = f(ijk, \phi) = 3, \quad i, j, k = 1, 2, 3 \; i \neq j \neq k$

$f(\phi, \phi) = 0$, then f is a bisubmodular function.

Definition 1.2.6. *Let $\mathcal{I} \subseteq D(E)$. $D = (E, \mathcal{I})$ is said to be a ditroid if the family $\mathcal{I}$ satisfies the following axiom.*

1)$\phi \in \mathcal{I}$

2) if $X \in \mathcal{I}$ and $Y \subseteq X$ then $Y \in \mathcal{I}$.

3) if X and $Y \in \mathcal{I}$ are non cancelling with $|X| = |Y| + 1$, then there exists an $e_i \in E$ and $Z \in \mathcal{I}$ such that

$Y(e_i) = 0$, $Z(e_i) = X(e_i) \neq 0$ and $Z(e_j) = Y(e_j)$ for all $j \neq i$.

The rank function of ditroid $D = (E, \mathcal{I})$ is a set function $h : D(E) \to \mathcal{R}_+$, defined by $h(X) = h(X_1, X_2) = max\{Y : Y \subseteq X$ and $Y \in \mathcal{I}\}$ for $X \in D(E)$.

Note: The rank function of a ditroid polyhedron is an increasing bisubmodular function.

Definition 1.2.7. *Let $\mathcal{O} \subseteq D(E)$. $M' = (E, \mathcal{O})$ is said to be a oriented matroid, if $\mathcal{O}$ satisfies the following properties.*

(1) For all $X \in \mathcal{O}$, $X \neq \phi$ implies $-X \in \mathcal{O}$, $\bar{Y} \subseteq \bar{X}$ implies $X = \pm Y$.

(2) For all $X = (X_1, X_2)$ and $(Y_1, Y_2) \in \mathcal{O}$, such that $X \neq -Y$ and for all $e \in (X_1 \cap Y_2) \cup (X_2 \cap Y_1)$ and $e' \in (X_1/Y_2) \cup (X_2/Y_1)$, there exists $Z = (Z_1, Z_2) \in \mathcal{O}$, such that $Z_1 \subseteq (X_1 \cup Y_1)/e$, $Z_2 \subseteq (X_2 \cup Y_2)/e'$ and $e' \in \bar{Z}$.

Graph theoretical Notations and Definitions:

A **directed graph** G=(V,A) consists of a set V of n nodes and a set A of m arcs whose elements are ordered pairs of distinct nodes.

A **directed network** is a directed graph whose nodes and/or arcs have associated numerical values (typically,costs, capacities, and/or supplies and demands). We often make no distinction between graphs and networks. Let us use the term graphs and networks synonymously. A undirected graph is defined in a same way as a directed graph/network except that the arcs in an undirected graph are unordered pairs of distinct nodes.

A directed arc (i,j) has two end points i and j. Node i has been referred to as the tail of the arc (i,j) and node j as its head. An arc (i,j) is incident to nodes i and j.

An arc (i,j) is an outgoing arc of node i and an incoming arc of node j. The **forwardstar** F(i) of node i is the set of arcs emanating from node i and the **backwardstar** B(i) is the set of arcs terminating at node i.

Let us assume that are no loops and multi arcs.

A **walk** in a directed graph G=(V,A) is a subgraph of G consisting of a sequence of nodes and arcs $i_1 - a_1 - i_2 - a_2 ... i_{r-1} - a_{r-1} - i_r$ satisfying the property that for all $1 \leq k \leq r-1$ either $a_k = (i_k, i_{k+1}) \in A$ *or* $a_k = (i_{k+1}, i_k) \in A$. A directed walk is a walk in which every two consecutive nodes i_k *and* i_{k+1} are connected by the arc (i_k, i_{k+1}).

A **path** is a walk in which no node is visited more than once.

A **cycle** is a path $i_1 - i_2 ... i_r$ together with the arc (i_r, i_1) *or* (i_1, i_r)

An arc (i,j) in a path/cycle is a forward arc if the path/cycle visits node i priori to visiting node j,and is a backward arc otherwise.

A directed path/cycle is a path/cycle in which every arc is a forward arc.

A graph is **acyclic** if it contains no directed cycle. Two nodes i and j are said to be connected if the graph contains at least one path from i to j. A graph is connected if every pair of its nodes is connected. Otherwise the graph is disconnected.

Minimum cost Network flow problem

Let us now present a mathematical programming formulation of the minimum cost flow problem (MCF). Let G=(V,A) be a directed network defined by a set V of n nodes and a set A of m directed arcs. Each arc (i,j) $\in$ A has an associated cost c_{ij} that denotes the cost per unit flow on that arc. We assume that flow cost varies linearly with the amount of flow. The amount of flow on each arc (i,j) $\in$ A is bounded below by l_{ij} and bounded above by u_{ij}. With each node $i \in V$, is associated an integer number b(i) representing its

supply/demand. If b(i)>0 ,node i is a supply node. If b(i)<0 then node i is a demand node. If b(i)=0 then it is a transshipment node. Flow on arc (i,j) $\in$ A is represented by x_{ij}.

The optimization model is as follows.

$$\min \sum_{(i,j)\in A} c_{ij}x_{ij}$$

$$\text{subject to } \sum_{j\,(i,j)\in A} x_{ij} - \sum_{j\,(j,i)\in A} x_{ij} = b(i)\ \forall i \in V$$

$$l_{ij} \leq x_{ij} \leq u_{ij} \forall (i,j) \in A$$

$$\text{where } \sum_{1}^{n} b_i = 0.$$

In matrix form, the minimum cost flow problem is as follows.

Minimize cx

subject to Nx=b

$l \leq x \leq u$, where N is a n x m matrix called the node-arc incident matrix.

Minimum cost submodular flow problem

Let us now present a mathematical programming formulation of the minimum cost submodular flow problem.

Let G =(V,A) be a directed graph with node set V and arc set A. Multiple arcs are allowed but loops are not.

A set $B \subseteq V$ is said to be a $(i\bar{j})$ set if $i \in B$ *and* $j \notin B$. An arc (i,j) leaves(enters) $B \subseteq V$, if B is $i\bar{j}$-set ($\bar{i}j$-set). Define a modular function λ_x as follows.

$$\lambda_x(B) = \sum_{(i,j) \text{ enters B}} x_{ij} - \sum_{(i,j) \text{ leaves B}} x_{ij}$$

Where $x \in \Re^E$ and $B \subseteq V$

Let $c \in \Re^E$, that is c is a real weighting of the arcs. Let U $\in (0, \infty)$ denote the flow upper bound vector. With out loss of generality lower bounds can be assumed to be zero. Let f be a crossing submodular function on the crossing families of subsets of V.

The mathematical formulation of the mincost submodular flow problem is as follows

Minimize cx

$$\text{subject to } \lambda_x(B) \leq f(B), \quad B \subseteq V$$

$$0 \leq x \leq U$$

A feasible solution x to the above problem is called a submodular flow and the set of all feasible solution is a submodular polyhedron.

Shortest path problem

The problem is to find a path of minimum cost (or length) from a specified node s to another specified node t, assuming that each arc (i,j) $\in$ A has an associated cost (or length) c_{ij}. It is a special case of the minimum cost flow problem.

1.3 Out line of the thesis

In **Chapter-2** an attempt has been made to study the properties of the submodular functions having non zero values at the empty set. When the value of f at the empty set is nonnegative the greedy algorithm will yield an optimal solution to the Linear programming problem (LPP) associated with $P(f)$. In this chapter it has been shown that the greedy algorithm may fail to solve the LPP when the submodular function has negative value at the empty set. We examine the reasons why the greedy algorithm fails. An algorithm to

solve the LPP when $f(\phi) < 0$, using minimizing the submodular function as an oracle is given. Some examples where submodular functions with negative values at the empty set occur are given . It has been shown that the system

$\{max\ cx : x \in P(f),\ f(\phi) < 0\}$ is also totally dual integral. Connectedness property has been so for defined for the submodular systems having $f(\phi) = 0$. We make an attempt to extend it to a general submodular systems. We also show that for a submodular function with $f(\phi) < 0$, it is possible that $P(f) \neq \phi$ but $B(f)$ the corresponding base polyhedron is empty. Lovasz [42] gave a method for extending a set function to all of $\mathcal{R}^n$ and proved that the extension is convex if and only if the set function is submodular. This result also requires that the set function must be zero at the empty set. We modify this result for functions with $f(\phi) < 0$.

In **Chapter-3** properties of bisubmodular functions with negative values at the empty set have been studied. The relationship between greedy algorithm and bisubmodularity was first pointed out by Kabadi and Chandresekeran [12]. They have shown that when $f(\phi, \phi) = 0$, the greedy algorithm implies bisubmodularity and vice versa. In this chapter an attempt has been made to study the relationship between bisubmodularity and the greedy algorithm when $f(\phi, \phi) \neq 0$. The question we ask is: Will bisubmodularity $\Rightarrow$ validity of the greedy algorithm? The answer is "not always".

The LPP associated with bisubmodular polyhedron can be solved by the generalised greedy algorithm when the value of f at empty set is nonnegative. But when $f(\phi, \phi) < 0$ the generalised greedy algorithm may fail to produce even a feasible solution. The LPP associated with a bisubmodular function having negative value at the empty set has been studied in detail. We have shown that it is not easy to solve the LPP associated with such functions. However, we give an efficient algorithm for solving the LPP,

$\{max\ cx :\ x \in \mathcal{R}^n,\ x(A,B) \leq f(A,B),\ (A,B) \subseteq (X,Y)\}$

where $X = \{e_i \mid c_{e_i} \geq 0\}$, $Y = \{e_i \mid c_{e_i} < 0\}$ and $f(\phi, \phi) < 0$.

Some of the existing theory for bisubmodular functions assumed to have zero value on empty sets is extended to functions having nonzero values at the empty set.

In **Chapter-4** we study the bisubmodular systems.

First we prove some properties of increasing bisubmodular functions and their associated polyhedra. We then extend these concepts to a general bisubmodular function and its associated polyhedron. We have shown that many properties of the polyhedron associated with increasing bisubmodular function and polymatroid are similar.

It has been shown that the tight sets corresponding to the non zero components of $v \in P(f)$ are non-cancelling. This property helps us to define support function, closure function and the partial order for the extreme points, when f is increasing.

We prove the Min-Max theorem for the bisubmodular system, and our proof differs from the existing ones. Relationship between separation problem and bisubmodular function minimization is also given. At the end we minimise bisubmodular functions with respect to some simple constraints, that occur in practical problems.

In Chapter-5 we develop a variant of cycle-cancelling algorithm for solving minimum cost network and submodular flow problems. The cycle-cancelling algorithm is credited to Klein [39]. Polynomial and strongly polynomial variants of this algorithm was developed by Goldberg and Tarjan [30], Barhona and Tardos [6] and Wallacher and Zimmerman [55]. Our cycle-cancelling algorithm for the minimum cost network flow performs the following steps on the residual network. We look for augmenting cycles as well as dynamically update potentials of the nodes blocking the search. The basic idea is to reduce the optimality violation of all arcs incident to the node blocking the search by half. The important feature of our algorithm is that it simultaneously finds negative cycles and tightens optimality viola-

tions by using the basic graph search algorithm. The complexity of our algorithm matches the complexity of Sokkalingam [52]. This algorithm has also been extended to solve the submodular flow problems. A study on computational behaviour of our algorithm has also been carried out in this chapter. Computational experimentation has been the primary method for comparative evaluation of network flow algorithms. We compare our algorithm with the relaxation algorithm for solving minimum cost network flow problem by Bertsekas and Tseng [8]. The test problems used are the standard bench marks generated by the publically available netgen program [40].

At the end we give concluding remarks, and highlight scope for future research.

Chapter 2

On Submodular functions having non zero values at the empty set

2.1 Introduction

Submodular functions with non-zero values on empty set occur while determining partitions of a submodular function $f - \lambda$ where λ varies from $-\infty$ *to* $+\infty$ Such partitions are required while analysing decomposition of electrical networks [44]. Such functions also occur when submodular functions defined on intersecting and crossing families are extended to the family of all subsets. Also parametric analysis of capacity constraints in submodular flow problems gives rise to submodular functions having non-zero values on empty set.
We show that for submodular functions with negative values on empty set the greedy algorithm may fail to solve the associated LPP. Also the nestedness property of the associated polyhedron may fail to hold. We give a efficient algorithm for solving an LPP over the polyhedra defined by such submodular functions using an algorithm for minimizing a submodular function as a oracle. We also study the total dual integrality, Lovasz extension and connectedness of such functions.

Definition 2.1.1. *Let* $E=(e_1, e_2...e_n)$ *be a finite set and let* f *be a submodular function defined on* D*, where* D *is the collection of all subsets of* E*.*
Let $p(f)$ *be the submodular polyhedron associated with* f*.*

Through out this chapter we will assume that an oracle is available which returns the value of $f(A)$ for any set $A \subseteq E$.

2.1.1 Linear programming problem LPP

For a submodular polyhedron $P(f)$ linear programming problem is defined as follows:

$$\begin{aligned} (P) \quad & Maximum \quad \sum_{i=1}^{n} c_i x_i \\ & \text{subject to} \quad x \in P(f) \end{aligned} \tag{2.1.1}$$

where $c \in \mathcal{R}_+^n$ is a weight vector corresponding to the elements of E.

The dual of problem (2.1.1) is

$$\begin{aligned} (D) \quad & Minimum \quad \sum_{A \in 2^E} f(A) Y_A \\ & \text{subject to} \\ & \sum_{e_i \in A \in 2^E} Y_A \quad = c_i, \forall e_i \in E, Y_A \geq 0, \forall A \in D \end{aligned} \tag{2.1.2}$$

If $f(\phi) = 0$, a greedy algorithm as in [19] for solving (2.1.1) is as follows.

Order the elements of E according to $c_1 \geq c_2 ... \geq c_n$. The vector $x \in \mathcal{R}^n$ solves (2.1.1) if $x_i = f(e_1, e_2 ... e_i) - f(e_1, e_2, ... e_{i-1}) \ \forall \ i = 1, ... n$.

The greedy solution vector x also represents an extreme point of the polyhedron $P(f)$.

In case $f(\phi) \neq 0$, it is not necessary that the greedy algorithm will solve (2.1.1).

In fact, the case, $f(\phi) \in (0, \infty)$ creates no problem for the greedy algorithm, but if $f(o) \in (-\infty, 0)$ then the greedy algorithm may not even obtain a feasible solution.

Example 1. *Consider the following example.*

Let E=(1,2,3) and f be a submodular function defined on subsets of E. Consider the linear programming problem defined on $P(f)$ as follows:

$$
\begin{array}{llllll}
Max & x_1 + x_2 + x_3 & & & & (2.1.3) \\
x_1 & \leq 3 & x_1 + x_2 & \leq 6 & & \\
x_2 & \leq 2 & x_2 + x_3 & \leq 8 & & \\
x_3 & \leq 4 & x_1 + x_3 & \leq 5 & & \\
f(\phi) & = -2 & x_1 + x_2 + x_3 & \leq 8 & &
\end{array}
$$

The greedy algorithm will produce the point (5,3,2) as the solution to (2.1.3), but we see that it is not a feasible solution as it violates the constraints $x_1 + x_2 \leq 6$,$x_1 + x_3 \leq 5$ and $x_1 + x_2 + x_3 \leq 8$.

It is easy to verify that the point (5,3,2) is an optimal solution to the problem with f replaced by $\bar{f}$, where $\bar{f}(A) = f(A) - f(\phi)\ \forall A \subseteq E$

But one still does not obtain the optimal solution (3,2,2) for the problem (2.1.3), since the two polyhedra $P(f)$ and $P(\bar{f})$ are totally different.

In what follows, we investigate the reasons for which greedy algorithm does not work for the case when $f(\phi) < 0$

Definition 2.1.2. *Let $P(f)$ be a submodular polyhedron and $x \in P(f)$, then a set A is said to be tight with respect to x or x-tight if $x(A) = f(A)$*

Fact: Let f be a submodular function and $f(\phi) = 0$ Then $A \cup B$ and $A \cap B$ are x-tight whenever A,B are.

Lemma 2.1.1. *Let $f(\phi) < 0$ and $x \in P(f)$. Let A,B be two x-tight sets. $A \cup B$ and $A \cap B$ are x-tight if $A \cap B \neq \phi$. If $A \cap B = \phi$ then the union and intersection of tight sets need not be tight.*

Proof. Since $x \in P(f)$ and A, B are x-tight $x(A) = f(A)$ and $x(B) = f(B)$.

$$\Rightarrow x(A \cup B) + x(A \cap B) = x(A) + x(B) = f(A) + f(B) \tag{2.1.4}$$

But,

$$f(A) + f(B) \geq f(A \cup B) + f(A \cap B) \tag{2.1.5}$$

If $A \cap B = \phi$ and $f(\phi) < 0$, it is possible that

$f(A) + f(B) < f(A \cup B)$

$\Rightarrow x(A \cup B) = x(A) + x(B) = f(A) + f(B) < f(A \cup B)$ and

$x(A \cap B) > f(\phi)$. Thus $A \cup B$ and $(A \cap B)$ are not x-tight sets.

If $A \cap B \neq \phi$ then it follows from (2.1.4)and (2.1.5) that $A \cup B$ and $A \cap B$ are both x-tight. □

Note: In case $f(\phi) > 0$, it follows from the above lemma, that $(A \cap B)$ and $(A \cup B)$ will both be x-tight sets.

Definition 2.1.3. *A (0,1)-matrix B is said to be nested if its columns can be permuted to obtain a matrix $\bar{B}$ that satisfies the condition*

$$\bar{B}_{ij} = 1 \text{ implies } \bar{B}_{ik} = 1 \quad \forall i, j \text{ and } \forall k \leq j.$$

Definition 2.1.4. *Linear programming problem (2.1.1) is said to have nestedness property if for any $c \in \mathcal{R}^E$ such that the dual (2.1.2) has an optimal solution, there exist an optimal dual solution y^* such that the row submatrix of the coefficient matrix of (2.1.2) corresponding to the positive components of y^* is nested.*

Definition 2.1.5. *Given a polyhedral set $P = \{x \cdot Ax \leq b\}$, $\bar{P} \subseteq P$ is a face of P if there exists a vector c such that*

$max\{c^t x : x \in P\}$ is attained precisely over $\bar{P}$.

Theorem (2.1.1) below relates nestedness property of the LPP (2.1.1) with submodularity of the right hand side function f.

Theorem 2.1.1. *(Kabadi,R.chandrasekaren [37])*

For the LPP (2.1.1), when $f(\phi) = 0$, the following statements are equivalent.

(1) (2.1.1) has the nestedness property.

(2)Greedy procedure yields an optimal solution to (2 1.1) $\forall$ $c \geq 0$.

(3) The function f is submodular.

Lemma 2.1.2. *Theorem (2.1.1) does not hold when $f(\phi) < 0$.*

Proof. We reproduce the proof from [37] to show that even when $f(\phi) < 0$, (1) $\Rightarrow$ (2) and (2) $\Rightarrow$ (3).

(1) $\Rightarrow$ (2) If necessary in the LPP(2.1.1) perturb the elements of c so that all elements are distinct. Let $\bar{P}$ be a minimal face of the polyhedron $P(f)$ over which the maximum value of the objective function (2.1.1) is attained. Let D denote the nested active matrix corresponding to the set of constraints defining $\bar{P}$. From duality theory of linear programming we know that there exists $y \geq 0$ such that y is optimal for the dual and $D^t y = 0$. Since D is nested it follows that $c_1 \geq c_2 \geq ..c_n$, and the greedy algorithm produces an optimal solution for (2.1.1).

(2) $\Rightarrow$ (3) For any $A, B \subseteq E$, define a vector c as

$$c_i = \begin{cases} 2 & if\ e_i \in (A \cap B), \\ 1 & if\ e_i \in (A \cup B) - (A \cap B), \\ 0 & otherwise \end{cases}$$

Let x^* denotes a greedy solution to (2.1.1). By definition of the greedy procedure

$x^*(A \cap B) = f(A \cap B)$, if $(A \cap B) \neq \phi$

=0 otherwise and

$x^*(A \cup B) = f(A \cup B)$.

It thus follows that

$$f(A) + f(B) \geq x^*(A \cup B) + x^*(A \cap B)$$
$$= f(A \cup B) + f(A \cap B) \text{ if } (A \cap B) \neq \phi$$
$$= f(A \cup B) + 0, \text{ if } (A \cap B) = \phi$$

Hence $f(A) + f(B) \geq f(A \cup B) + f(A \cap B)$, that is f is a submodular function.

It has already been shown through example (1) that submodularity of f does not guarantee that greedy procedure yield an optimal solution and that corresponding LPP have the nestedness property. So (3) $\not\Rightarrow$ (2) and (3) $\not\Rightarrow$ (1) □

Consider the dual of the LPP in example (1)

$\min (3y_1 + 2y_2 + 4y_3 + 6y_4 + 8y_5 + 5y_6 + 8y_7)$

Subject to

$y_1 + y_4 + y_6 + y_7 = 1$

$y_1 + y_4 + y_5 + y_7 = 1$

$y_1 + y_5 + y_6 + y_7 = 1$

$y_i \geq 0$

The optimal solution is

$y^* = (0.1, 0, 0, 0, 1, 0)$ and the coefficient matrix corresponding to y^* is

$$\begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}$$

which is not a nested matrix.

2.1.2 Algorithm for solving LPP over $P(f)$, when $f(\phi) \neq 0$

For the LPP defined in (2.1.1), the greedy algorithm will also obtain an optimal solution in case $f(\phi) > 0$, since one can always treat the value of f at empty set as zero. However it is not possible to do so when $f(\phi) < 0$.

For solving the problem (2.1.1) with $f(\phi) < 0$ we assume that an oracle for minimizing a submodular function is available with complexity (k). we also need the following results. With out loss of generality let as assume that the value of f at all singletons are zero. If the given function f does not satisfy this property then convert f into another function $\bar{f}$ so that it satisfies the above property. One can do this by subtracting a modular function w, where $w(e) = f(e)$ for every $e \in E$.

That is for any $A \subseteq E$, let $\bar{f}(A) = f(A) - \sum_{e_i \in A} w(e_i)$

Lemma 2.1.3. *Solving the LPP over $P(f)$ is equivalent to solving it over $P(\bar{f})$.*

Proof. This is just a translation of $P(f)$ by a vector w=$\{f(e_1), f(e_2), ...f(e_n)\}$ and hence from the definition of $\bar{f}$ it follows that if $x \in P(\bar{f})$, then $x + w \in P(f)$.

$\Rightarrow$ $P(\bar{f}) = P(f) - \{w\}$,

Thus if $(x_1, x_2...x_n)$ is an optimal solution for an LPP over $P(\bar{f})$, then $\{x_1 + w(e_1), x_2 + w(e_2), ...x_n + w(e_n)\}$ will be an optimal solution for the same LPP over $P(f)$. □

From now on we will refer to LPP (2.1.1) with the function f having the property that it has zero values on singletons.

Lemma 2.1.4. *To find a non − empty minimising set of a submodular function f, the oracle for minimising f may be used atmost $|E|$ times.*

Proof. Apply the oracle for minimising f and if the empty set is the minimising set then, let $E' = E - \{i\}$

Define a function f_i on E' as follows

$f_i(A) = f(A \cup i)$ $\forall$ $A \subseteq E'$

f_i is submodular.

Let A_i be the minimising set for f_i and let

$f_k(A_k) = \min_i(f_i(A_i))$. Then A_k is the non-empty minimising set for f. □

Let us first solve a simpler case of the LPP (2.1.1) where c's are all 1. Then by using this algorithm we can solve (2.1.1) for any $c \geq 0$ We use the approach in [44] to solve problem (2.1.1), when all c_i's =1. Intialy find the set A that minimizes f. If $f(A) > 0$, the right hand side vector in (2.1.1) is non-negative and the constraints $x_i \leq 0$ for singletons

$$\Rightarrow max \sum_{i=1}^{n} x_i = 0$$

and hence $x_i = 0 \ \forall i$ is the optimal solution.

If $f(A) < 0$, since $f(B) \geq f(A) \ \forall B \subseteq A$, it follows that $max \sum x_i = f(A)$. We now treat the set A as a singleton and repeat the above process for the set $E' = (E/A) \cup \{A\}$, that is replace f by f' on E' such that it has zero values on singletons, and again find the non-empty minimising set with respect to E'. Repeat this process until $E = \phi$ or the minimum value of the current submodular function is non-negative.

We now give a concise version of our algorithm.

The following procedure will find a non empty minimising set A with respect to some subset $B \subseteq E$, for a submodular function f such that $f(A) < 0$.

Procedure $(Min - Neg(f, E))$

Begin

set $(B = \phi)$

Step 1: If e $\in E/B$ then $B = B \cup$ e

else go to Step 3

Step 2: $f(A) = \min_{\phi \subset X \subseteq B} f(X)$

If $f(A) < 0$ then return A

else go to Step 1.

Step 3: Return (ϕ)

end

In Step-2 one has to use the oracle for minimising a submodular function together with lemma (2.1.4).

Note 1. *If the procedure Min-Neg returns ϕ at the first iteration itself then $Max \sum x_i = 0$, hence $x_1 = x_2 = ... = x_n = 0$.*

Main Algorithm

i=0;

Begin

Step 1: i=i+1;

A_i=CALL (Min-Neg(f, E))

If $(A_i = \phi)$ then go to Step 3

else go to Step 2

Step 2: Set $E' = (\{A_i\} \cup E/(A_i))$

(Treat the set A_i as a singleton in E'. So $\mid E' \mid = \mid E/A_i \mid +1$.

Now define a submodular function f' on E' such that $f'(A_i) = 0$.

that is consider a modular function w on E' such that

$w([A]) = f(A_i)$ and $w(e) = f(e)$ $\forall e \in E/A_i$)

Now $f' = f - w$, f' is submodular and $f'(A_i) = 0$

set $f = f'$ and $E = E'$

go to Step 1

Step 3: STOP

end.

This algorithm terminates only when the procedure $Min - Neg(f, E)$ returns the empty set.

At the end of the algorithm we will have

$$E' = ([A_1], [A_2], ..[A_k], (E/ \cup_{i=1}^{k} A_i)),$$

$$x_e = 0 \ \forall e \in (E/\cup_{i=1}^{k} A_i) \quad \text{and}$$

$$\sum_{e_i \in A_1} x_i = f(A_1), \ldots \sum_{e_i \in A_k} x_i = f(A_k)$$

So

$$Max \sum_{e_i \in E} x_i = \sum_{i=1}^{k} f(A_i) + 0.$$

which is the optimum value.

The validity of the algorithm follows easily. We first show that the vector x obtained at the end of the algorithm is in $P(f')$, where the ground set

$$E' = ([A_1], [A_2], ..[A_k], (E/\cup_{i=1}^{k} A_i)).$$

Let A be any subset of E'. The algorithm terminates when, for every subset of E', the corresponding value of $f' \geq 0$. Also for the singletons in E', the corresponding f' value is 0. Therefore $\forall i, x_i \leq 0$.

Thus $\sum_{e_i \in A} x_i \leq 0 \leq f'(A) \Rightarrow \ x \in P(f')$

If $w_1, ..., w_k$ are the k-modular functions introduced during the course of the algorithm, then from lemma-2.1.3, it is clear that $x + w_1 + ... + w_k \ \in P(f)$.

As for the outcome x being an optimal solution, suppose there exist a $x^* \in P(f')$ such that

$$max \sum_{e_i \in E} x_i^* > \sum_{e_i \in E} x_i$$

Since the value of f' at singletons are zero it follows that

$$\sum_{e_i \in E/\cup_{i=1}^{k} A_i} x_i^* \leq 0$$

therefore for some A_j

$$\sum_{e_i \in A_j} x_i^* > \sum_{e_i \in A_j} x_i = f'(A_j) = 0$$

that is $x^*(A_j) > 0$ which is a contradiction. Therefore the algorithm obtains an optimal solution. We can now solve the LPP (2.1.1) for any $c \geq 0$.

Using the above algorithm, replace f in (2.1.1) by $\bar{f}$ where

$$\bar{f}(A) = \left(Max \sum_{i \in A} x_i : \quad x \in P(f) \right) \forall A \subseteq E. \tag{2.1.6}$$

$\bar{f}$ is submodular and its value at the empty set is zero.

Lemma 2.1.5. $P(f) = P(\bar{f})$

Proof. Let $x \in P(f)$ we show that x $\in P(\bar{f})$

$x \in P(f)$

$\Rightarrow x(A) \leq f(A) \forall A \in 2^E$,But

$$\bar{f}(A) = Max \sum_{i \in A} x_i : x \in P(f)$$

So

$$Max \sum_{i \in A} x_i \geq \sum_{i \in A} x_i = x(A)$$

$\Rightarrow x(A) \leq \bar{f}(A)$

$\Rightarrow x \in P(\bar{f}), \quad$ so $\quad P(f) \subseteq P(\bar{f})$

The converse part is of course obvious.

So $P(f) = P(\bar{f})$. □

We can now apply the greedy algorithm to solve (2.1.1) over the polyhedron $P(\bar{f})$.

Note that after ordering the c_i's as $c_1 \geq c_2 \geq ... \geq c_n$, we need to calculate $\bar{f}$ for the n sets $\{(e_1), (e_1, e_2)...(e_1, e_2, ...e_n)\}$ only.

Complexity

Complexity of finding a non-empty minimising set requires the oracle for finding a minimum set atmost $|E|$ times. To find $(Max \sum x_i : x \in P(f))$ we saw that the procedure min-neg(f,E) has to be used atmost n-times. Thus the complexity of solving problem (2.1.1) when all c_i's =1 is $o(n^2k)$. Finally to solve (2.1.1) for arbitrary $c \geq 0$, we use the greedy

algorithm. This would require calculation of $\bar{f}$ for n sets, thus the total complexity of our algorithm for solving LPP (2.1.1) is $o(n^3k)$.

2.2 Examples

In this section we discuss some problems where submodular functions with $f(\phi) < 0$ occur.

Extension of submodular functions

In [42] Lovasz considers the extension of submodular functions defined on intersecting and crossing families to the family of all the subsets. We show that such extensions will give rise to submodular functions with negative values on empty set

Let $\mathcal{F}$ be a intersecting family with $\phi \notin \mathcal{F}$.

Let $S_1, S_2 ... S_k$ be the maximal members of the intersecting family $\mathcal{F}$

Now for $\forall X \subseteq E$, $X \neq \phi$, $\mathcal{F}$ contains a unique smallest subset containing X. Let it be denoted by $\overline{X}$.

Define

$$f_i(X) = \begin{cases} f\left(\overline{X \cap S_i}\right) & \text{if } X \cap S_i \neq \phi \\ -c & \text{if } X \cap S_i = \phi \end{cases}$$

Where $\overline{X \cap S_i}$ is the unique smallest set in $\mathcal{F}$ that contains $(X \cap S_i)$. Now the extension of f is defined as follows

$$\bar{f}(X) = \sum_{i=1}^{k} f_i(X) + (k-1)c \quad \forall\, X \subseteq E$$

where c is a large number.

Now $\bar{f}(\phi)$=-kc +kc -c= -c < 0.

Extensions of submodular functions defined on crossing families will also lead to submodular functions having negative values on empty set.

Parameterization of submodular flow constraints

We will define the submodular flow problem as follows:

Given a digraph G(V,E) with node set V and arc set E

$$\min \sum c_{ij} x_{ij} \qquad (2.2.1)$$

subject to

$$\lambda_x(A) < f(A) \quad A \subseteq V$$

$$\text{where} \quad 0 \leq x \leq g \quad \text{and } \lambda_x(A) = \sum_{j \in A} x_{ij} - \sum_{i \in A} x_{ij}$$

f is a crossing submodular function on a crossing family F of subsets of V and $g \in (R \cup \infty)^E$.

A feasible solution to (2.2.1) is called a submodular flow. Here $f(\phi) = 0$.

The following is a well known result in [27].

Theorem 2.2.1. *If f is a submodular function on a crossing family F with $\phi, V \in F$ and $f(\phi) = 0$ the corresponding polyhedron $P(f)$ is*

$$P(f) = (x/x \in \mathcal{R}^V, \ \forall X \in F \quad x(X) \leq f(X))$$

If $P(f) \neq \phi$ then there exits a unique submodular function f' such that $P(f) = P(f')$, and f' is defined over all subsets of V. Also $f'(\phi) = 0$. More over if f is integer valued so is f'.

Thus in problem (2.2.1) we can replace f by a fully submodular function f'. Parametrisation of the flow constraints will give rise to submodular functions $f' - \lambda$, $\lambda \in (-\infty, \infty)$. These functions will have non-zero values on the empty set.

LPP and Partition problems

Let f be a submodular function with $f(\phi) = 0$. Now consider the function $\bar{f} = f - \lambda$, where $\lambda \in (-\infty, \infty) \cup \Re$. Clearly $\bar{f}$ is a submodular function and $\bar{f}(\phi) = f(\phi) - \lambda = -\lambda$.

First consider the case when $\lambda \in (-\infty, 0]$, the LPP

$$Max\left(\sum cx : x \in P(\bar{f})\right) \tag{2.2.2}$$

can be solved by the greedy procedure.

Lemma 2.2.1. *Suppose* $(x_1, x_2, ...x_n)$ *is the greedy solution for problem (2.2.2) for* $\lambda = 0$ *then* $(x_1 - \lambda, x_2...x_n)$ *is an optimal solution for all* $\lambda \in (-\infty, 0]$.

Proof. Define $g(A) = \bar{f}(A)\ \forall \phi \neq A \subseteq E$ and $g(\phi) = 0$.

It can be immediately verified that $P(g) = P(\bar{f})$.

We now apply the greedy algorithm to (2.2.2) with $\bar{f}$ replaced by g, and let x' denote the outcome of the greedy algorithm, then

$x'_1 = g(1) - g(\phi) = f(1) - \lambda - 0 = x_1 - \lambda$

$x'_2 = g(1,2) - g(1) = (f(1,2) - \lambda) - (f(1) - \lambda) = x_2$

$\vdots$

$x'_n = g(1,2...n) - g(1,2...n-1) = (f(1,2...n) - \lambda) - (f(1,2...n-1) - \lambda) = x_n$ □

The case $\lambda \in (0, \infty)$, has already been discussed earlier and we have shown that the point $(x_1 - \lambda, x_2..., x_n)$ may not be a feasible solution for (2.2.2) in this case.

Partition problem

In electrical networks analysis by decomposition [44] we come across the problem of finding a partition π of E, that minimises

$$\overline{f - \lambda}(\pi) = \sum_{N_i \in \pi} \overline{f - \lambda}(N_i)$$

That is finding the

$$\min_{\pi}\left(\overline{f - \lambda}(\pi)\right)$$

where f is a submodular function on the subsets of E with $f(\phi) = 0$ and π is a partition of E and N_i's are the components of the partition π and λ takes values between $-\infty$ and ∞.

consider the dual of problem (2.2.2) with $c_i = 1\ \forall i$

$$min \sum_A y(A)(f(A) - \lambda)$$

$$\text{subject to } \sum_{e_i \in A} y(A) = 1 \quad i = 1, ...n.$$

The optimal solution to the dual will give the minimum partition of E with respect to the submodular function $(f - \lambda)$ and solutions to the dual as λ varies from $(-\infty, \infty)$, will give the principle partition lattice [44] of the submodular function f

Note 2. *(a) For $\lambda \in (-\infty, 0) \Rightarrow \bar{f}(\phi) \geq 0$. In the earlier section we have shown that the LPP(2.2.2) can be solved by the greedy algorithm for all $\bar{f}$ for which $\bar{f}(\phi) \geq 0$. Thus for the case when all c_i's=1 from linear programming duality the optimal solution to the dual problem will be y(E)=1 and all other $y_A = 0$, $A \subseteq E$. That is, for all $\lambda \in (-\infty, 0)$, E itself is the minimum partition for $f - \lambda$.*

(b) For $\lambda \in (0, \infty)$, $\bar{f}(\phi) < 0$ and the primal problem corresponding to the partition problem can be solved by the algorithm discussed in the previous section. During the course of this algorithm we also obtain the minimum partition of $f - \lambda$.

2.3 Properties of submodular functions with $f(\phi) < 0$

Base Polyhedron

Let $P(f) = \left(x \in \Re^E : x(A) \leq f(A) \quad \forall\ A \in 2^E\right)$ be the polyhedron associated with f.

Let us define the base polyhedron as

$B(f) = \left(x \in \Re^E : x \in P(f) \quad \text{and} \quad x(E) = f(E)\right)$

If $f(\phi) = 0$ then the base polyhedron$B(f)$ will be nonempty whenever$P(f)$ is nonempty.

But this is not true in the case when $f(\phi) < 0$ It is possible that the base polyhedron is empty even though $P(f)$ in non empty

Suppose there exists two sets A,B such that $A \cap B = \phi$ and $A \cup B = E$ and

$f(A) + f(B) < f(A \cup B) = f(E)$

Then $\forall x \in P(f)$

$x(A) + x(B) \leq f(A) + f(B) < f(E)$

this implies that

$$\sum_{i \in A \cup B} x_i < f(E) \; \forall x \in P(f)$$

Thus $B(f)$ may be empty, even if $P(f)$ is non-empty.

That is one cannot always define the base polyhedron for a submodular function with $f(\phi) < 0$

Definition 2.3.1. *Let f be a submodular function on the subsets of E with $f(\phi) = 0$. Now define a function f^* on the subset system $(E/X, X \subseteq E)$ as follows:*

$f^*(E/X) = f(E) - f(X)$.

Clearly f^* is supermodular and $f^*(\phi) = 0$.

In [27] it has been shown that $B(f) = B(f^*)$ when ever $f(\phi) = 0$

Consider the above statement with $f(\phi) < 0$. In this case the statement need not be true. Since $f(\phi) < 0$, $B(f)$ may be empty. But since $f^*(\phi) = 0$ for any submodular function f, $B(f^*) \neq \phi$.

Hence $B(f)$ may not be the same as $B(f^*)$ when $f(\phi) < 0$

Submodular Function Minimisation

An oracle for minimising a submodular function is an integral part of most of the combinatorial optimization problems defined on a submodular polyhedron.

In case $f(\phi) < 0$, we can apply the existing algorithms for minimising a submodular function for which $f(\phi) = 0$, to the function $f - f(\phi)$, since the minimising set for $f - f(\phi)$ is also the minimising set for f.

The Min-Max theorem for f submodular, when $f(\phi) = 0$ is as follows:

$$\max\left(y(E) : \ y \in P(f), \ y \leq 0\right) = \min\left(f(X) : \ X \in 2^E\right) \tag{2.3.1}$$

For proving this result the property that union and intersection of tight sets are also tight is used. Since this property does not always hold when $f(\phi) < 0$, the above Min-Max theorem may not always be valid for the case $f(\phi) < 0$.

Replace f by the function $\bar{f} = f - f(\phi)$ in the equation (2.3.1) and the corresponding min-max theorem is

$$\max\left(y(E) : \ y \in P(\bar{f}), \ y \leq 0\right) = \min\left(\bar{f}(X) : \ X \in 2^E\right)$$

Suppose minimum on the right-hand side occurs for a set $A \subseteq E$, as mentioned earlier A will also be the minimising set for f.

Total Dual Integrality

Theorem 2.3.1. *The system* $(Max \sum c_i x_i : x \in P(f), f(\phi) < 0)$ *is totally dual Integral.*

Proof. Define a function $\bar{f}$ as follows:

$\bar{f}(A) = (Max \sum x_i : x \in P(f))$

By the earlier lemma one can see that $P(f) = P(\bar{f})$ and the system

$(Max \sum c_i x_i : x \in P(\bar{f}))$ is totally dual Integral. So the given system is totally dual integral. Thus the TDI property is not affected when $f(\phi) < 0$. □

Theorem 2.3.2. *Let* f, g *be any two integer valued submodular function with* $f(\phi)$ *and* $g(\phi) < 0$. *Let* $P(f)$ *and* $P(g)$ *be the corresponding submodular polyhedron then* $P(f) \cap P(g)$ *is TDI.*

Proof. Let $\bar{f}(A) = \max\{\sum x(A) : x \in P(f)\}$

$\bar{g}(A) = \max\{\sum x(A) \cdot x \in P(g)\}$

Now $P(\bar{f}) = P(f)$ *and* $P(\bar{g}) = P(g)$

So $P(\bar{f}) \cap P(\bar{g}) = P(f) \cap P(g)$

Now $\bar{f}(\phi) = \bar{g}(\phi) = 0$, so $P(\bar{f}) \cap P(\bar{g})$ is TDI by [42] and hence $P(f) \cap P(g)$ is TDI. □

Connectedness

In this section we investigate the connectedness for a submodular system (f, D) when $f(\phi) < 0$.

Definition 2.3.2. *Let (f, D) be any submodular system on E with $f(\phi) = 0$*
We say that the submodular system is connected if there is no nonempty proper subset X of E such that

$$f(X) + f(E - X) = f(E)$$

Definition 2.3.3. *Let $B(f)$ be the base polyhedron, then for any $x \in B(f)$,*
let $\mathcal{F} = \{X \mid X$ is x-tight set$\}$. Define $A_x^e = \{\cap X \mid e \in X,\ X \in \mathcal{F}\}$
Define a graph G(x)=(E,A(x)), where E is the vertex set, and $A(x) = \{(e, e') \mid e' \in A_x^e\}$.
If (f, D) is connected then the graph G(x) is connected for any $x \in B(f)$.

We now examine connectedness of the submodular system (f, D) when $f(\phi) < 0$. Definition (2.3.2) is no longer valid in this case.

For example consider the following submodular system (f, D), where

$E = (e_1, e_2, e_3)$, $f(e_i) = 2$, i=1to 3, $f(e_1, e_2) = 6$, $f(e_2, e_3) = 8$, $f(e_1, e_3) = 5$,

$f(e_1, e_2, e_3) = 6$ and $f(\phi) = -4$.

No proper subset X of (e_1, e_2, e_3) satisfies the condition

$f(X) + f(E - X) = f(E)$. Thus this system (f, D) is connected.

But the graph G(2,2,2) is not connected as shown below.

$x = (2, 2, 2) \in B(f)$ and $\mathcal{F} = \{(e_1), (e_2), (e_3), (e_1, e_2, e_3)\}$

$A_x^{e_1} = \{(e_1)\}$, $A_x^{e_2} = \{(e_2)\}$ and $A_x^{e_3} = \{(e_3)\}$

$\Rightarrow$ $A(x) = (\phi)$, and hence G(x) is a disconnected graph.

So. the definition (2.3.2) may not hold for a submodular system with $f(\phi) < 0$.

We will define the connectedness as follows:

Let (f, D) be a given submodular system with $f(\phi) < 0$ Define a submodular function $\bar{f}$ such that

$\bar{f}(A) = (max\ x(A)\ :\ x \in P(f))$

Definition 2.3.4. *Let (f, D) be any submodular system on E with $f(\phi) < 0$ We say that the submodular system (f, D) is connected if there is no nonempty proper subset X of E such that*

$$\bar{f}(X) + \bar{f}(E - X) = \bar{f}(E)$$

In the theorem below we give necessary and sufficient conditions for a submodular system to be connected in terms of the corresponding base polyhedron.

Theorem 2.3.3. *Let (f, D) be any submodular system with $f(\phi) < 0$ and $B(\bar{f})$ be the corresponding base polyhedron of $\bar{f}$. The submodular system (f, D) is connected if and only if there exists a base $x \in B(\bar{f})$ such that $x(X) < \bar{f}(X)$, for all $X \in D$ with $E \neq X$.*

Proof. Let $x \in B(\bar{f})$ such that $x(X) < \bar{f}(X)\ \forall X \in D, X \neq E$, then

$\bar{f}(E) = x(E) = x(X) + x(E - X) < \bar{f}(X) + \bar{f}(E - X)$ for all X $\subset$ E and $X \neq E$.

$\Rightarrow$ (f, D) is connected.

Conversely let (f, D) be connected. To prove that there exist a base $x \in B(\bar{f})$ such that

$x(X) < \bar{f}(X)\ \forall X \in D\ X \neq E$

Suppose not, then for any $x \in B(f)$ there exits a set Y $\in D$ such that

$x(Y) = \bar{f}(Y)$ *and* $x(E - Y) = \bar{f}(E - Y)$, $Y \neq E, \phi$.

(If $x(E-Y) < \bar{f}(E-Y)$, then one can decrease the values of x_i's $\in$ Y and make $x(Y) < \bar{f}(Y)$)

Now, $\bar{f}(E) = x(E) = x(Y) + x(E-Y) = \bar{f}(Y) + \bar{f}(E-Y)$.

this contradicts our assumption that (f, D) is connected. Therefore there exists a base $x \in B(\bar{f})$ such that $x(X) < \bar{f}(X)$, for any X $\in$ D with $E \neq X$. □

Thus one can define the connectedness for any general submodular system (f, D).

The Lovasz Extensions of submodular functions

Lovasz extends a submodular function f to $\mathcal{R}_+^n$ by defining

$$f'(c) = \max\left(cx \mid x \in P(f) \ \forall \quad c \in \mathcal{R}_+^E\right)$$

Since f is submodular and χ_T denotes the characteristic vector of the subset T of E, then by the greedy algorithm when $f(\phi) = 0$, $f'(\chi_T) = f(T)$.

When $f(\phi) < 0$ $f'(\chi_T) \leq f(T)$ and in some cases the inequality may be strict.

From the thoery of linear programming one knows that $f'(c)$ is convex, where f is any orbitary set function $\forall c \in \mathcal{R}^n$ and hence $\forall c \in \mathcal{R}_+^n$.

But convexity of f' need not imply submodularity of f since f is not the restriction of f' to (0,1) vectors.

Restriction of f' to (0,1) vectors is however a submodular function.

We now try to answer the reverse question. that is when is the restriction of a convex function to (0,1) vectors a submodular function? Lovasz raised this problem in [42].

Consider the convex function $f(x_1, x_2, x_3) = x_1^2 + x_2^2 + x_3^2 + x_1x_2 + x_1x_3 + x_2x_3$ *on* $\mathcal{R}^3$

It can be immediately verified that restriction of f to (0,1) vectors does not yield a submodular function. Consider the polyhedron $P(f)$ obtained by restricting f to (0,1) vectors.

If for any $X \subseteq E$ and a convex function f, $f(\chi_X) = max\ (\chi_X(y) :\ y\ \in P(f))$

then f restricted to (0,1) vectors will be a submodular function.

Chapter 3

On Bisubmodular functions having non zero values at the empty set

3.1 Introduction

Bisubmodular functions with non-zero values at the empty set occur while parameterising the right-hand side vector of the LPP over a bisubmodular polyhedron. Such functions also occur when bisubmodular function defined on intersecting families are extended to the family of all subsets. Also parametric analysis of the capacity constraints in the bisubmodular flow problems give rise to bisubmodular functions having non zero values at the empty set.

We show that for bisubmodular functions with negative values at the empty set the greedy algorithm fails to solve the associated LPP. Also the nestedness property of the associated polyhedron may fail to hold. We also study in detail the LPP over the polyhedron defined by such bisubmodular functions. We also study some more properties of such functions and their associated polyhedra.

Definition 3.1.1. *Let E be the finite set and $D(E)$ be the collection of directed sets of E. let f be a bisubmodular function from $D(E) \rightarrow \mathcal{R}$.*
Let $P(f)$ be the Bisubmodular Polyhedron associated with f.

We now characterise the non-emptiness of the bisubmodular polyhedron $P(f)$, when $f(\varphi, \phi) < 0$. A necessary and sufficient condition for $P(f)$ to be non-empty when $f(\phi, \phi) = 0$ as in [34] is that $f(A, B)+f(B, A) \geq 0 \; \forall (A, B) \in D(E)$. A natural modification of the above condition for the case $f(\phi, \phi) < 0$ would be that it is satisfied for all $(A, B) \neq (\phi, \phi)$. This is what we state in theorem (3.1.1) The proof is the same as in [34]. We reproduce it here for completeness sake.

Theorem 3.1.1. *Let f be a bisubmodular function with $f(\phi, \phi) \leq 0$, Its associated polyhedron $P(f)$ is non empty if and only if $\forall (A, B) \in D(E)$ and $(A, B) \neq (\phi, \phi)$.*
$f(A, B) + f(B, A) \geq 0$

Proof. Let $P(f) \neq \phi$ and let $x \in P(f)$
$\Rightarrow f(A, B) + f(B, A) \geq x(A, B) + x(B, A) = 0$ (Since $(A, B) \neq (\phi, \phi)$).
Conversely, Let $f(A, B) + f(B, A) \geq 0$
Let us prove this by induction.
For $|E|$=1. Then
$f(e, \phi) + f(\phi, e) \geq 0$,
that is, $f(e, \phi) \geq -f(\phi, e)$
Choose $x(e) \in (-f(\phi, e), f(e, \phi))$
Then $x(e) \in P(f)$. Hence the result is true for $|E|$=1 Let us assume that it is true for $|E|$=k-1.
Let us prove for $|E|$=k.
Define a bisubmodular function f_1 on $E' = E - s$, for some $s \in E$ as follows:
$f_1(A, B) = f(A, B) \; \forall (A, B) \in D(E')$. Clearly f_1 is bisubmodualr on $D(E')$ and $|E'|$=k-1.
So by induction there exists $x \in P(f_1)$.

Let

$$m = \min_{(A,B)\in D(E)} \{f(A,B) - x(A,B) \cdot s \in A\}$$

$$M = \max_{(A,B)\in D(E)} \{x(A,B) - f(A,B) \,.\, s \in B\}$$

Clearly $M \leq m$.

Choose $x(s) \in (M, m)$.

$x^* = x \in P(f_1)$ *and* $x(s) \in (M, m)$.

$\Rightarrow x^* \in P(f)$. □

Through out this chapter we will assume that $P(f) \neq \phi$, $P(f)$ is bounded and an oracle is available which will return the value of $f(A,B)$ for any set (A,B) $\in D(E)$.

3.1.1 Linear programming problem (LPP)

For a bisubmodular polyhedron $P(f)$, linear programming problem is defined as,

$$(P) \qquad \max \quad \left(\sum_{e_i\in E} c_i x_i\right) \qquad (3.1.1)$$

Subject to $x \in P(f)$

where $c \in \mathcal{R}^n$ is a weight vector corresponding to the elements of E.

Its dual can be written as:

$$(D) \qquad \min \sum_{(A,B)\in D(E)} f(A,B) Y_{(A,B)} \qquad (3.1.2)$$

Subject to

$$\sum_{\substack{(A\,B)\in D(E)\\ e_i\in A}} Y_{(A,B)} - \sum_{\substack{(A.B)\in D(E)\\ e_i\in B}} Y_{(A,B)} = c_i \;\forall e_i \in E,$$

$$Y_{(A,B)} \geq 0 \;\forall (A,B) \in D(E)$$

If $f(\phi,\phi) = 0$, a generalised greedy algorithm for solving (3.1.1) is as follows.

Order the elements of E according to

$P = \{i_1, i_2, ...i_n : |c_{i_1}| \geq |c_{i_2}| \geq ... \geq |c_{i_n}|\}$

The vector $x \in \mathcal{R}^n$ solves (3.1.1), where

$$x_i = \begin{cases} f(A_i, B_i) - f(A_{i-1}, B_{i-1}) & if\ c_i \geq 0 \\ f(A_{i-1}, B_{i-1}) - f(A_i, B_i) & if\ c_i < 0 \end{cases}$$

the indices i being ordered as in P and,

$$\begin{aligned} (A_0, B_0) &= (\phi, \phi) \\ (A_i, B_i) &= (A_{i-1}, B_{i-1} \cup e_i) \quad if\ c_i \geq 0 \\ &= (A_{i-1} \cup e_i, B_{i-1}) \quad if\ c_i < 0 \end{aligned}$$

The vector x also represents an extreme point of the polyhedron $P(f)$.

In case $f(\phi, \phi) \neq 0$, it is not necessary that the greedy algorithm will solve (3.1.1).

In fact, the case $f(\phi, \phi) \in (0, \infty)$ creates no problem for the greedy algorithm, but if $f(\phi, \phi) \in (-\infty, 0)$, then the greedy algorithm may not even obtain a feasible solution.

In what follows, we investigate the reasons for which the generalised greedy algorithm does not work when $f(\phi, \phi) < 0$.

For the LPP (3.1.1) form the diset (X,Y) such that

$X = \{e_i : c_i \geq 0\}$ and $Y = \{e_i : c_i < 0\}$

So the objective function in LPP (3.1.1) can be written as

$$\max\left(\sum_{e_i \in E} c_i x_i\right) = \max\left(\sum_{e_i \in X} c_i x_i - \sum_{e_i \in Y} c_i x_i\right)$$

Definition 3.1.2. *Let f be a bisubmodular function and $P(f)$ be its associated polyhedron. Let $x \in P(f)$. A set (A,B) is said to be a x-tight set if $x(A, B) = f(A, B)$.*

Fact: [46] Let f be a bisubmodular function and $f(\phi, \phi) = 0$, then union and intersection of any two x-tight sets are x-tight.

Lemma 3.1.1. *Let f be a bisubmodular function with $f(\phi, \phi) < 0$ and $x \in P(f)$. If (A,B) and (C,D) are any two x-tight sets, then $(A, B) \cup (C, D)$ and $(A, B) \cap (C, D)$ need not be x-tight.*

Proof. Since $x \in P(f)$ and (A,B), (C,D) are x-tight sets $x(A,B) = f(A,B)$ and $x(C,D) = f(C,D)$ and hence,

$$x(A \cup C \backslash B \cup D, B \cup D \backslash A \cup C) + x(A \cap C, B \cap D) = x(A,B) + x(C,D) = f(A,B) + f(C, \quad (3.1$$

But,

$$f(A,B) + f(C,D) \geq f(A \cup C \backslash B \cup D, B \cup D \backslash A \cup C) + f(A \cap C, B \cap D) \qquad (3.1.4)$$

If $(A,B) \cap (C,D) = \phi$ and $f(\phi,\phi) < 0$, it is possible that $f(A,B) + f(C,D) < f(A \cup C \backslash B \cup D, B \cup D \backslash A \cup C)$

$$\begin{aligned} \Rightarrow x(A \cup C \backslash B \cup D, B \cup D \backslash A \cup C) &= x(A,B) + x(C,D) \\ &= f(A,B) + f(C,D) \\ &< f(A \cup C \backslash B \cup D, B \cup D \backslash A \cup C) \end{aligned}$$

and $x((A,B) \cap (C,D)) > f(\phi,\phi)$.

Thus $(A,B) \cup (C,D)$ and $(A,B) \cap (C,D)$ need not be x-tight. However when $(A,B) \cap (C,D) \neq \phi$ then by equations (3.1.3) and (3.1.4) we have union and intersection of x-tight sets are also x-tight. □

Definition 3.1.3. *A $(0, \pm 1)$ matrix B is said to be a signed nested matrix if it can be transformed into a nested (0,1) matrix by a ± 1 scaling of its columns.*

Definition 3.1.4. *Linear program (3.1.1) is said to have signed nested property if for any c in $\mathcal{R}^E$ such that (3.1.2) has an optimal solution. there exists an optimal solution y^* to (3.1.2) such that the row submatrix of the coefficient matrix of (3.1 2) corresponding to the positive components of y^* is signed nested.*

Theorem 3.1.2. *(Kabadi,R.chandrasekaren)*

If f is a bisubmodular function with $f(\phi,\phi) = 0$ then the following statements are equivalent:

(1)(3.1.1) has signed nested property.

(2)Generalised greedy procedure yields an optimal solution $\forall c \in \mathcal{R}^E$

(3) The function f *is bisubmodular.*

Lemma 3.1.2. *Theorem (3.1.2) does not hold when* $f(\phi, \phi) < 0$.

Proof. As in the submodular case one can show that even when $f(\phi, \phi) < 0$, (1) $\Rightarrow$ (2) and (2) $\Rightarrow$ (3)

Let x denote an optimal solution for (3.1.1). However, as discussed earlier, union and intersection of x-tight sets need not be x-tight and hence the row submatrix of the coefficient matrix of (3.1.2) corresponding to the positive components of x need not be signed nested and bisubmodularity of f need not guarantee that the greedy algorithm will yield an optimal solution to LPP (3.1.1). So (3) $\not\Rightarrow$ (2) and (3) $\not\Rightarrow$ (1) □

Consider the following example.

Example 2.

$$max(x_1 - x_2)$$

$$\begin{array}{rr} x_1 \leq 1 & x_1 - x_2 \leq 3 \\ -x_1 \leq 1 & -x_1 + x_2 \leq 0 \\ x_2 \leq 1 & x_1 + x_2 \leq 4 \\ -x_2 \leq 1 & -x_1 - x_2 \leq 2 \end{array}$$

$$f(\phi, \phi) = -2$$

The generalised greedy algorithm produces the solution (3,2) which is not a feasible point. The primal optimal solution is (1,-1) and the corresponding optimal solution of the dual problem is (1 0 0 1 0 0 0 0). The row submatrix of the coefficient matrix corresponding to the positive components of the optimal solution is

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 \end{pmatrix}$$

which is not a signed nested matrix.

Another important consequence of the value of f at empty set is as follows:

Consider the LPP given in (3.1.1) and the LPP

$$\max\left(\sum_{e_i\in X} c_i x_i - \sum_{e_i\in Y} c_i x_i\right) \tag{3.1.5}$$

Subject to $x(A,B) \leq f(A,B)\ \forall (A,B) \subseteq (X,Y)$

where $e_i \in X$ if $c_i \geq 0$ and $e_i \in Y$ if $c_i < 0$.

If $f(\phi,\phi) = 0$, from the generalised greedy algorithm it follows that

$$\begin{array}{lll} \max\left(\sum_{e_i\in X} c_i x_i - \sum_{e_i\in Y} c_i x_i\right) & = & \max\left(\sum_{e_i\in X} c_i x_i - \sum_{e_i\in Y} c_i x_i\right) \\ \text{Subject to} & & \text{subject to} \\ x(A,B) \leq f(A,B)\ \forall (A,B) \in D(E) & & x(A,B) \leq f(A,B)\ \forall (A,B) \subseteq (X,Y) \end{array} \tag{3.1.6}$$

However when $f(\phi,\phi) < 0$

$$\begin{array}{lll} \max\left(\sum_{e_i\in X} c_i x_i - \sum_{e_i\in Y} c_i x_i\right) & \leq & \max\left(\sum_{e_i\in X} c_i x_i - \sum_{e_i\in Y} c_i x_i\right) \\ \text{Subject to} & & \text{subject to} \\ x(A,B) \leq f(A,B)\ \forall (A,B) \in D(E) & & x(A,B) \leq f(A,B)\ \forall (A,B) \subseteq (X,Y) \end{array}$$

Since the optimal solution of the r.h.s problem need not be feasible for the l.h.s problem. The linear programme on the r.h.s of 3.1.6 can be formulated as an LPP over a submodular polyhedron as follows: For $A \subset X \cup Y$ define $f'(A) = f(A_1, A_2)$, where $A_1 = A \cap X$ and $A_2 = A \cap Y$. $f'(A)$ is a submodular function over the subsets of $X \cup Y$.

Let $c'_i = c_i\ \forall e_i \in X$

$= -c_i\ \forall e_i \in Y$

The LPP on r.h.s of 3.1.6 can be rewritten as

$$max\ \left\{\sum c'_i y_i\ :\ y(A) \leq f'(A)\ \forall A \subseteq X \cup Y\right\}$$

The case $f(\phi,\phi) > 0$ can be handled in the same way as for the submodular case in the previous chapter.

Solving problem (3.1.1) does not appear to be easy for the case $f(o.\phi) < 0$. However we give an efficient algorithm for solving the LPP, given in (3.1.5), defined on an orthant.

3.1.2 Algorithm

To solve problem (3.1.5) when $f(\phi,\phi) < 0$ we need the following results and definitions.

Lemma 3.1.3. *Let f be any bisubmodular function, defined on $D(E)$.*

$$\min_{(A,B)\subseteq(X,Y)}(f(A,B)), \quad \textit{where } (X,Y) \textit{ is given diset and } (X,Y)\in D(E)$$

is equivalent to finding

$\min_{A\subseteq E}(\bar{f}(A))$ *where* $E=(X\cup Y)$ *and* $\bar{f}$ *is a submodular function defined as follows*

$$\bar{f}(A) = f(A_1,A_2), \quad \textit{where } (A_1) = (A\cap X) \textit{ and } A_2 = (A\cap Y)$$

Proof. We will show that $\bar{f}$ is submodular, on subsets of $X\cup Y$

$$\begin{aligned}
\bar{f}(A)+\bar{f}(B) &= f(A_1,A_2)+f(B_1,B_2)\\
&\geq f(A_1\cup B_1\backslash A_2\cup B_2, A_2\cup B_2\backslash A_1\cup B_1)+f(A_1\cap B_1, A_2\cap B_2)\\
&= f(A_1\cup B_1, A_2\cup B_2)+f(A_1\cap B_1, A_2\cap B_2), Since\ A_1\cap B_2=\phi=A_2\cap B_1\\
&= \bar{f}(A\cup B)+\bar{f}(A\cap B)
\end{aligned}$$

$\Rightarrow \bar{f}$ is submodular, and by definition of $\bar{f}$, it follows that

$$\min_{A\subseteq E=(X\cup Y)}\bar{f}(A) = \min_{(A_1,A_2)\subseteq(X,Y)} f(A_1,A_2)$$

Hence the proof. □

Note: We have already proved in chapter-2 that, to find a non-empty minimising set of a submodular function f, the oracle for minimising f may be used atmost $|E|$ times.

Definition 3.1.5. *A bisubmodular function f is said to be a zero bisubmodular function with respect to a given diset (X,Y), if*

$f(e,\phi) = 0 \quad and \quad f(\phi,e) = 0 \quad \forall (e,\phi)$ *or* $(\phi,e) \in (X,Y)$

Definition 3.1.6. *A function g is said to bimodular if*

$$g(A,B) + g(C,D) = g(A \cup C \backslash B \cup D, B \cup D \backslash A \cup C) + g(A \cap C, B \cap D)$$

$\forall (A,B),(C,D) \in D(E)$

Lemma 3.1.4. *If g is a bimodular function and if $P(g) \neq \phi$, then $g(\phi,\phi) \geq 0$.*

Proof. Let (A,B) be any non empty diset.

g bimodular $\Rightarrow g(A,B) + g(B,A) = g(\phi,\phi) + g(\phi,\phi)$

If $g(\phi,\phi) < 0$, then $g(A,B) + g(B,A) < 0$

Since $P(g) \neq \phi$, by theorem (3.1.1),

$g(A,B) + g(B,A) \geq 0$ $\forall (A,B) \neq (\phi,\phi)$ Thus $g(\phi,\phi) \geq 0$. □

Let us assume that f is a zero bisubmodular function with respect to a given diset (X,Y). If not, then convert the given bisubmodular function to the zero bisubmodular function by subtracting a bimodular function w which is defined as follows.

$$\forall (e,\phi) \in (X,Y) \left| \begin{array}{l} w(e,\phi) = f(e,\phi) \\ w(\phi,e) = -f(e,\phi) \end{array} \right.$$

$$\forall (\phi,e) \in (X,Y) \left| \begin{array}{l} w(\phi,e) = f(\phi,e) \\ w(e,\phi) = -f(\phi,e) \end{array} \right.$$

$w(\phi,\phi) = 0$

Clearly w is bimodular and $f - w$ is a zero bisubmodular function with respect to (X,Y).

Lemma 3.1.5. *Solving the LPP over $P(f)$ is equivalent to solving it over $P(f-w)$.*

Proof. From the definition of $f-w$ it follows that if $x \in P(f-w)$ then $x+w \in P(f)$, thus if, $(x_1, x_2 ... x_n)$ is an optimal solution for an LPP over $P(f-w)$ then $(x'_1, x'_2 ... x'_n)$ is an optimal solution for the same LPP over $P(f)$, where

$$x'_i = \begin{cases} x_i + w(i,\phi) & if\ (i,\phi) \in (X,Y) \\ x_i + w(\phi,i) & if\ (\phi,i) \in (X,Y) \end{cases}$$

□

From now on we will refer to the LPP (3.1.5) with f a zero bisubmodular function with respect to the given diset (X,Y).

Let us first solve the case where c_i's are ± 1.

Let $X = \{i : c_i = 1\}$ and $Y = \{i : c_i = -1\}$

Now the LPP is,

$$\begin{aligned} &Maximise && \left(\sum_{e_i \in X} x_i - \sum_{e_j \in Y} x_j \right) \\ &\text{Subject to} && x(A,B) \leq f(A,B) \text{ where } (A,B) \subseteq (X,Y) \end{aligned} \tag{3.1.7}$$

We extend the algorithm of the previous chapter for solving the LPP associated with submodular functions to the LPP associated with bisubmodular functions. Since minimising a bisubmodular function over a given diset is equivalent to that of minimising a submodular function by lemma (3.1.3) , we will assume that an oracle for minimising a submodular function is available to us. Modifying the diset (X,Y) at each iterations and maintaining the bisubmodular function as a zero bisubmodular function with respect to the modified diset (X,Y) at each iterations are the main steps involved in the algorithm.

For a given zero bisubmodular function f with respect to (X,Y), define a submodular function f' as in lemma (3.1.3).

Intialy find a non-empty minimum set A' of E such that $f'(A') < 0$ and for sets of B_i's, $\subseteq A'$ such that $\cap B_i = \phi$ and $\cup B_i = A'$, $f'(A') \leq \sum f'(B_i)$. If there exist no such set then

$$max \sum_{i \in X} x_i - \sum_{j \in Y} x_j = 0$$

and hence $x_i = 0 \ \forall i$ is the optimal solution.

If A' is the set with above property and (A,B) is the corresponding diset of A', then set

$$max \sum_{A} x_i - \sum_{B} x_j = f(A, B)$$

Set $E' = \{(E - A'), (A')\}$. Treat the set A' as singleton in E'. Let $A' = e'$ say. So $E' = (E - A') \cup e'$.

Let w be a modular function such that $(f' - w)$ is a zero submodular function on E'. Set $E = E'$ and $f' = f' - w$ and repeat the above process until there exists no set A' on E with the above mentioned property. We now give the concise version of our algorithm.

Procedure (Min-Neg(f,E))

(This procedure is there in chapter-2)

The procedure Min-Neg(f,E) will find a non empty minimising set (A') for a submodular function f having the above mentioned property. If there is no such set then it will return an empty set.

Note 3. *If the procedure Min-Neg(f,E) returns (ϕ) at the first iteration itself then*

$$Max \left(\sum_{i \in X} x_i - \sum_{j \in Y} x_j \right) = 0 \quad where \quad x_1 = x_2 = \ .. = x_n = 0.$$

We now give the main algorithm for solving the LPP (3.1.7). f' refers to the submodular function obtained from f with respect to the orthant (X,Y).

Main Algorithm

i=0

Begin

Step 1: i=i+1;

(A^i) = **Call Min-Neg** (f', E)

If $(A^i) = (\phi)$ then go to step 4.

Step 2: Let the corresponding diset of A^i be (A_i, B_i)

(Find the new set E' and the function f' as discussed earlier)

Step 3: Go to step 1 with the new set E' and f';

Step 4: Stop

end

At the end of the algorithm we will have the following partitions of (X,Y).

$$(X, Y) = (A_1, B_1) \cup (A_2, B_2)...(A_k, B_k) \cup (X \setminus \cup_{i=1}^{k} A_i, Y \setminus \cup_{i=1}^{k} B_i)$$

Now

$$\forall (e, \phi) \quad or \quad (\phi, e) \in (X/\cup_{i=1}^{k} A_i, Y/\cup_{i=1}^{k} B_i) \quad , x_e = 0$$

and the optimum value is

$$Max \left(\sum_{i \in X} x_i - \sum_{j \in Y} x_j \right) = \sum_{i=1}^{k} f(A_i, B_i)$$

Validity of the algorithm has already been proved in the previous chapter for the sub-modular case.

We can now solve the LPP (3.1.5) for any $c \geq 0$.

Using the above algorithm, replace f in (3.1.5) by $\bar{f}$ where,

$$\bar{f}(A, B) = max \left(\sum_{i \in A} x_i - \sum_{i \in B} x_i \mid x(C, D) \leq f(C, D) \; \forall \; (C, D) \subseteq (A, B) \right)$$

The following lemma is obvious

Lemma 3.1.6. *Solving the LPP (3.1.5) with the function f is equivalent to that of solving it with $\bar{f}$*

One can apply the generalised greedy algorithm to solve (3.1.5) with the function $\bar{f}$. For that we need to calculate the value of $\bar{f}$ only at n disets, which can be found by the above algorithm.

Complexity

The complexity of the algorithm is the same as for the submodular case dealt in Chapter-2. Thus our algorithm will solve LPP 3.1.5 in $o(n^3k)$, where k is the complexity for minimising a submodular function.

From equation (3.1.6), one can see that if x' is an optimal solution to the LPP 3.1.5 and if $x' \in P(f)$, then x' is also an optimal solution to (3.1.1). If $x' \notin P(f)$ then, solving LPP (3.1.1) is not easy. However checking that $x' \in P(f)$ will require solving a membership problem for $P(f)$. We show in the next chapter that the membership problem is equivalent to minimising a bisubmodular function. If $x' \notin P(f)$, since the solution vector x of LPP (3.1.1) is not necessarily an extreme point of the polyhedron associated with LPP(3.1.5) it is not clear how the knowledge of x' can be used to obtain x.

3.2 Examples

In this section we discuss some problems where bisubmodular functions with $f(\phi, \phi) < 0$ occur.

Extension of bisubmodular functions

In this we will show that bisubmodular functions with negative values on empty sets occur in the extension of bisubmodular function defined on intersecting families $\mathcal{F}$ to all of $D(E)$.

Let $\mathcal{F}$ be a intersecting family with $(\phi,\phi) \notin \mathcal{F}$.

Let $(A_1,B_1),(A_2,B_2)...(A_k,B_k)$ be the maximal members of the intersecting family $\mathcal{F}$.

Now for $\forall (X,Y) \subseteq D(E)$, $(X,Y) \neq (\phi,\phi)$, $\mathcal{F}$ contains a unique smallest subset including (X.Y). Let it be denoted by $\overline{(X,Y)}$.

Define

$$f_i(X,Y) = \begin{cases} f\left(\overline{(X,Y)\cap(A_i,B_i)}\right) & \text{if } (X,Y)\cap(A_i,B_i) \neq (\phi,\phi) \\ -c & \text{if } (X,Y)\cap(A_i,B_i) = (\phi,\phi) \end{cases}$$

Now the extension is defined as follows

$$\bar{f}(X,Y) = \sum_{i=1}^{k} f_i(X,Y) + (k-1)C \quad \forall\, X \subseteq E$$

where c is a large positive number.

Now $\bar{f}(\phi,\phi)$=-kc +kc -c= -c.

Here this bisubmodular function has negative value on the empty set.

Partition Problem

Partitions of directed sets

Definition 3.2.1. *We say that* $((A_1,B_1),(A_2,B2)...(A_k,B_k))$ *is a partition of (A,B) if* $(A_i,B_i) \in D(E) \forall i$, $(A_i \cap B_i) = \phi$ *and* $\cup_{i=1}^{k}(A_i,B_i) = (A,B)$

Let π denotes the collection of all partitions of (A,B)

Let us define

$$\bar{f}(A,B) = \min_{\pi} \sum_{i=1}^{k} f(A_i,B_i)$$

Here the minimum is taken over all the partitions of (A,B)

Now consider the dual of the LPP (3.1.7)

$$(D) \quad \textit{Minimum} \quad \sum y(A,B) f(A,B)$$

$$\textit{subject to} \quad \sum_{i\in X} y(A,B) = 1$$

$$\sum_{i\in Y} y(A,B) = -1, \quad y(A,B) \geq 0 \; \forall \; (A,B) \subseteq (X,Y)$$

Here any partition of (X, Y) is a feasible solution to the dual problem.

Note that the set of all minimum partitions of (X.Y) forms a lattice.

The following lemma is obvious.

Lemma 3.2.1. *If $f(\phi, \phi) = 0$ then,*

$$Minimum \sum y(A, B) f(A, B) = f(X, Y)$$

Consider the above dual problem with f replaced by f' where $f' = f - \lambda$ and $\lambda \in (0, \infty)$, then f' is a bisubmodular function having negative value at the empty set

Note: $P(f')$ can be empty by theorem-(3.1.1), but one can always find a partition of a given diset (X,Y) with respect to f' by using the algorithm given in the previous section.

3.3 Properties of BSM functions with $f(\phi, \phi) < 0$

Total dual Integrality

Let f be a bisubmodular function. Define a function f' as follows:

$$f'(A, B) = max\{x(A, B) : x \in P(f)\} \tag{3.3.1}$$

When $f(\phi, \phi) = 0$, $f'(A, B) = f(A, B)$

If $f(\phi, \phi) < 0$, $f'(A, B) \leq f(A, B)$, but the following claim can be made.

$P(f) = P(f')$ and f' is a bisubmodular function. Again since $P(f)$ is bounded f' is always finite.

Theorem 3.3.1. *Let f be a bisubmodular function with $f(\phi, \phi) < 0$, then the system of inequalities $\{maxcx : x \in P(f)\}$ is TDI.*

Proof. By the above claim, $max\{cx : x \in P(f)\} = max\{cx : x \in P(f')\}$

Since f' is bisubmodular and $f'(\phi, \phi) = 0$ it follows from the greedy algorithm that r.h.s has the TDI property.

Since $P(f) = P(f')$, it follows that the left hand side also has the TDI property. □

Base polyhedron

Let $(f, D(E))$ be any given bisubmodular system and (X,Y) be any orthant in D(E). Define the polyhedron $P_{(X,Y)}(f)$ as follows:

$$P_{(X,Y)}(f) = \left\{x | x \in \mathcal{R}^E, \forall (A,B) \subseteq (X,Y) : x(A,B) \leq f(A,B)\right\} \tag{3.3.2}$$

Now we will define the base polyhedron of the orthant (X,Y) as,

$$B_{(X,Y)}(f) = \left\{x : x \in P_{(X,Y)}(f), x(X,Y) = f(X,Y)\right\} \tag{3.3.3}$$

One can see that this base polyhedron $B_{(X,Y)}(f)$ has the same combinatorial structure as the base polyhedron of a submodular system. Hence when $f(\varphi, \varphi) < 0$, the base polyhedron $B_{(X,Y)}(f)$ can be empty even when $P_{(X,Y)}(f)$ is non empty.

When $f(\phi, \phi) \geq 0$, to check $B_{(X,Y)}(f)$ is empty or not it is enough to check $P_{(X,Y)}(f)$ is empty or not. However it is not the case when $f(\phi, \phi) < 0$.

Lemma 3.3.1. *Let f be any bisubmodular function with $f(\phi, \phi) < 0$, then $B_{(X,Y)}(f)$ is non empty if and only if $P_{(X,Y)}(f)$ is nonempty and there exist no partition $(A_i, B_i)_{i=1}^k$ of (X,Y) such that*

$$\sum_{i=1}^{k} f(A_i, B_i) < f(X,Y)$$

Proof. If $P_{(X,Y)}(f)$ is non-empty and if there exist a partition $(A_i, B_i)_{i=1}^k$ of (X,Y) such that

$$\sum_{i=1}^{k} f(A_i, B_i) < f(X,Y)$$

then for every $x \in P_{(X,Y)}(f)$

$$x(X,Y) = x(\cup_1^k (A_i, B_i) \leq \sum_{i=1}^{k} f(A_i, B_i) < f(X,Y)$$

$\Rightarrow x(X,Y) < f(X,Y)$ and hence the base polyhedron is empty. □

Connectedness

Let $(f, D(E))$ be any bisubmodular system. with $f(\phi, \phi) = 0$ For any given orthant (X.Y) define $P_{(X,Y)}(f)$ and $B_{(X,Y)}(f)$ as in equations (3.3.2) and (3.3 3).

Let $f_{(X,Y)}$ be the submodular function corresponding to the polyhedron $P_{(X,Y)}(f)$.

That is for $A \subseteq E$ $f_{(X,Y)}(A) = f(A_1, A_2), where A_1 = A \cap X, A_2 = A \cap Y$

We have already shown that $P_{(X,Y)}(f) = P(f_{(X,Y)})$ and $B_{(X,Y)}(f) = B(f_{(X,Y)})$

Definition 3.3.1. *Let $(f, D(E))$ be any bisubmodular system with $f(\phi, \phi) = 0$ and $P(f) \neq \phi$. $(f, D(E))$ is said to be connected if for any orthant (X, Y), the submodular system $(f_{(X,Y)}, D)$ is connected,*

that is there exists no non empty proper subset A of E such that

$f_{(X,Y)}(E - A) + f_{(X,Y)}(A) = f_{(X,Y)}(E)$

In other words

for any given orthant (X, Y) there exists no non empty proper subset A of E such that

$f(X \cap A, Y \cap A) + f(X/A, Y/A) = f(X, Y)$.

Let us now extend the concept of connectedness to a bisubmodular system $(f, D(E))$ with $f(\phi, \phi) < 0$. When $f(\phi, \phi) < 0$, then from lemma (3.3.1) it follows that $B_{(X,Y)}(f)$ can be empty and hence it is not possible to define the connectedness for the submodular system $(f_{(X,Y)}, D)$. To avoid this we define a function f' as in equation (3.3.1).

Clearly $f'(\phi, \phi) = 0$ and $f'_{(X,Y)}$ is a submodular function with respect to the orthant (X.Y).

Definition 3.3.2. *Let $(f, D(E))$ be a bisubmodular system with $f(\phi, \phi) < 0$. We will say that $(f, D(E))$ is connected if for any orthant (X, Y) the submodular system $(f'_{(X,Y)}, D)$, where D is the collection of subsets of E is connected,*

that is for any given orthant (X, Y) there exists no non empty proper subset A of E such that $f'(X \cap A, Y \cap A) + f'(X/A, Y/A) = f'(X, Y)$.

In the following lemma we characterise connectedness of $(f, D(E))$ when $f(\phi, \phi) < 0$, in terms of the function f'. Our proof of the lemma differs from that of [4].

Lemma 3.3.2. *Let $(f, D(E))$ be any connected system, with $f(\phi, \phi) < 0$ then there is no $(A, B) \in 3^E$ such that $f'(A, B) + f'(B, A) = 0$, $A \cup B \neq E, \phi$ and (A, B) , $(B, A) \in D(E)$.*

Proof. Since $(f, D(E))$ is connected, from our definition it follows that

for any orthant (X,Y) the submodular system $(f'_{(X,Y)}, D)$ is connected. By theorem (2.3.3) of chapter-2 there exists atleast one base $x \in B_{(X,Y)}(f')$ such that

$x(A) < f'_{(X,Y)}(A)\ \forall \phi \subset A \subset E$

$\Rightarrow\ x(A_1, A_2) < f'(A_1, A_2),\ \forall\ (\phi, \phi) \subset (A_1 A_2) \subset (X, Y)$

Let (A,B) be some diset such that $A \cup B \neq E, \phi$ and $f'(A, B) + f'(B, A) = 0$.

We have atleast one orthant (X,Y) such that $(A, B) \subset (X, Y)$.

Let $f'_{(X,Y)}$ be the submodular function corresponding to (X,Y) There exists atleast one $x \in P(f')$ such that $x \in B_{(X,Y)}(f')$ and

$x(A, B) < f'(A, B)$

$\Rightarrow\ x(A, B) + x(B, A) < f'(A, B) + f'(B, A) = 0.$

(since $(B, A) \in D(E)$ and $x(B, A) \leq f'(B, A)$.)

$\Rightarrow\ x(A, B) + x(B, A) < 0$, which is a contradiction.

So there is no (A,B) with $(A \cup B) \neq E, \phi$ such that $f'(A, B) + f'(B, A) = 0$

Hence the lemma. □

Definition 3.3.3. *A bisubmodular system $(f, D(E))$ with $f(\phi, \phi) < 0$ and $P(f) \neq \phi$ is said to be fully connected if there exists no $(\phi, \phi) \neq (A, B) \in 3^E$ such that $f'(A, B) + f'(B, A) = 0$, where $f'(A, B) = max(x(A) - x(B) \mid x \in P(f))$.*

That is we say that $(f, D(E))$ is fully connected if $(f', D(E))$ is fully connected.

Note 4. *Combining the criterion for non emptiness of $P(f')$ in theorem (3.1.1) and the definition for fully connectedness of $(f', D(E))$ it follows that $(f', D(E))$ is fully connected*

if $f'(A,B) + f'(B,A) > 0\ \forall(\phi,\phi) \neq (A,B) \in D(E)$.

Lemma 3.3.3. $(f, D(E))$ *is fully connected bisubmodular system with* $f(\phi,\phi) < 0$ *if and only if* $P(f)$ *is full dimensional.*

Proof. First we will show that, if $(f, D(E))$ is fully connected, then $P(f')$ is full dimensional.

Since $(f', D(E))$ is fully connected,

$f'(A,B) + f'(B,A) > 0\ \forall(\phi,\phi) \neq (A,B) \in D(E)$ by the above Note.

From the proof for theorem (3.1.1), for non emptiness of $P(f')$, it follows that the interval $[m,\ M]$ contains an interior point for all k. Hence $p(f')$ has an interior point.

So if $(f, D(E))$ is fully connected then $P(f')$ is full-dimensional

Since $f'(A,B) \leq f(A,B)\ \forall(A,B) \in D(E)\ (A,B) \neq (\phi,\phi)$ it follows that $P(f)$ is also full-dimensional.

Conversely if $P(f)$ is full dimensional we show that the bisubmodular system $(f, D(E))$ is fully connected. Since $P(f) = P(f')$, full dimensionality of $P(f)$ implies full dimensionality of $P(f')$. By our definition $f'(\phi,\phi) = 0$ and by Fujishige's result in [5], it follows that $(f', D(E))$ is fully connected. □

In the above section we have made an attempt to extend the concept of connectedness to the bisubmodular functions having $f(\phi,\phi) < 0$. Other results regarding connectedness such as defining connectedness for the restriction of f to some set A and obtaining a bidirected graph regarding connectedness can also be extended in a similar manner.

Bisubmodular function Minimisation

Bisubmodular function minimisation does not depend on the value of f on empty set. In case $f(\phi,\phi) < 0$, one can always convert f into another bisubmodular function f' where

$f'(A, B) = f(A, B) - f(o, \phi)$, and

$$\min_{(A,B)\in D(E)} f(A, B) = \min_{(A,B)\in D(E)} f'(A, B).$$

In chapter-4 we discuss the min-max theorem for a bisubmodular systems $(f, D(E))$ and show that this theorem is not valid when $f(\phi, \phi) < 0$.

Chapter 4

Bisubmodular Systems

4.1 Introduction

A bisubmodular polyhedron is defined in terms of so-called bisubmodular function on a family of ordered pairs of disjoint subsets of a finite set Bisubmodular systems were examined by Fujishige [4], L. QI [46] and guha et.el [33]. Rank functions of ditroids, perfect ditroids and oriented matroids are increasing bisubmodular functions. Again a g-polymatroid defined with respect to two increasing submodular functions is also a bisubmodular polyhedron associated with a increasing bisubmodular function.

In this chapter we make an attempt to study the structure of increasing bisubmodular functions and their associated polyhedra. We then extend these concepts to the general bisubmodular function and its associated polyhedron. We also prove the min-max theorem for the bisubmodular polyhedron in a way different from that of Cunningham [16] and investigate the relationship between the separation problem and bisubmodular function minimization.

At the end of the chapter we minimize a bisubmodular function with respect to some simple constraints that arises in some practical problems.

4.2 Increasing Bisubmodular functions

In [37] Chandrasekaren and Kabadi introduced the concept of pseudomatroids. They gave the polyhedral characterization of pseudomatroids as.

$x(A, B) \leq b(A, B)\ \forall (A, B) \in D(E)$

where $b : D(E) \rightarrow \mathcal{R}$ is an integer valued. bisubmodular function on D(E), satisfying the following conditions:

1)$b(\phi, \phi) = 0$

2)$b(e_i, \phi) = 0$ or $1\ \forall e_i \in E$

3)$\{A \subseteq C, D \subseteq B;\ (A, B), (C, D) \in D(E) \Rightarrow b(A, B) \leq b(C, D)$.

Note that here b is not an increasing bisubmodular function on D(E). In this section we study increasing bisubmodular function and the polyhedral characterization is given by.

$x(A, B) \leq f(A, B)\ \forall (A, B) \in D(E)$

where f is a bisubmodular function satisfying the following conditions:

(1) $f(\phi, \phi) = 0$

(2) f is increasing. That is,

$\{\forall (A, B), (C, D) \in D(E)$ such that$(A, B) \subseteq (C, D).\ f(A, B) \leq f(C, D)\}$

A study on increasing bisubmodular function will help us to characterize different kinds of bisubmodular polyhedra such as Ditroid polyhedron. g-polymatroid corresponding to two increasing submodular function and the polyhedron corresponding to the rank function of the oriented matroids.

We have shown that the polyhedron $P(f)$ associated with an increasing bisubmodular function has many properties similar to that of a polymatroid. We also show that tight sets corresponding to non-zero components of $v \in P(f)$ will be non-cancelling. Using this property we define the closure, support function and partial order for any $v \in P(f)$ in the same way as for a submodular function.

Definition 4.2.1. *Two disets (A,B) and (C,D) are said to be cancelling sets if* $(A \cap D) \cup (B \cap C) \neq \phi$

Definition 4.2.2. *An incremental function of a function* $f \quad D(E) \rightarrow \mathcal{R}$ *is defined as follows:*

$f_j(X) = f(X + j) - f(X)$ *where* $X = (X_1, X_2) \in D(E)$ *and we write* $X + e$ *in place of* $X \cup (e, \phi)$ *or* $X \cup (\phi, e)$ *It will always be clear from the context, whether e is a forward element or a backward element.*

Below we give two lemmas from [34] which characterise increasing bisubmodular functions in terms of their incremental function.

Lemma 4.2.1. *Let* $f : D(E) \rightarrow \mathcal{R}$ *be a non-decreasing function.* f *is bisubmodular if and only if its incremental function is non-increasing, ie if and only if,*

$f_j(X) = f(X + j) - f(X) \geq f(X + j + k) - f(X + k) = f_j(X + k)$,

for all $j, k \in E/\bar{X}$ $j \neq k$.

Lemma 4.2.2. f *is an increasing bisubmodular function if and only if for any two non-cancelling disets (A,B),(C,D)* $\in D(E)$,

$$f(A, B) \leq f(C, D) + \sum_{j \in A} (f(C \cup j, D) - f(C, D)) + \sum_{j \in B} (f(C, D \cup j) - f(C, D))$$

We now show that tight sets corresponding to non-zero components of $v \in P(f)$ are non-cancelling.

Theorem 4.2.1. *Let* f *be a increasing bisubmodular function and* $P(f)$ *be the polyhedron associated with it. For* $v \in P(f)$, *tight sets corresponding to non-zero components of* v *are non-cancelling.*

Proof. Let $v_e > 0$ and (A+e,B) and (C,D+e) be two v-tight sets That is,

$$v(A + e, B) = f(A + e, B) \tag{4.2.1}$$

$$v(C, D + e) = f(C, D + e) \tag{4.2.2}$$

Now from equation (4.2.1)

$$v_e + v(A, B) = f(A + e, B) \geq f(A, B) \text{ (since } f \text{ is increasing)}$$

$\Rightarrow v_e + v(A, B) \geq f(A, B)$

$\Rightarrow v_e \geq f(A, B) - v(A, B) \geq 0 \quad$ since $\quad v(A, B) \leq f(A, B)$

So $v_e \geq 0$

similarly from equation(4.2.2) we get

$-v_e + v(C, D) = f(C, D + e) \geq f(C, D)$

$\Rightarrow \ v_e \leq 0$

so $v_e = 0$. which is a contradiction.

Hence the proof. □

We can now define the support and closure functions for $v \in P(f)$

Definition 4.2.3. *Support function for $v \in P(f)$ is defined as follows*

$$sup(v) = \begin{cases} (e, \phi) \text{ if } v_e > 0 \\ \cup \\ (\phi, e) \text{ if } v_e < 0 \end{cases}$$

Since either (e, ϕ) or $(\phi, e) \in sup(v)$ for any $v \in P(f)$, $sup(v) \neq \phi$, for $v \neq 0$.

Definition 4.2.4. *Closure function for $v \in P(f)$ is defined as follows*

$cl(v) = \cup((A, B) \mid (A, B) \quad is \quad v - tight \quad set)$

Note: Since tight sets corresponding to non-zero component of v are non-cancelling one can always define the closure function as above.

Closure of v is always tight but support need not be.

Definition 4.2.5. *We can now define the partial order on the elements of cl(v).*

For (e, ϕ) or $(\phi, e) \in cl(v)$

$$(A, B)^v_{(e,\phi)} = (\cap(A, B) \mid \quad (e, \phi) \in (A, B) \quad where \quad (A, B) \quad is \quad v - tight)$$

$$(A,B)^v_{(\phi,e)} = (\cap(A,B) \mid \quad (\phi,e) \in (A,B) \quad where \quad (A,B) \quad is \quad v-tight)$$

Since f is increasing for any $e_1, e_2 \in cl(v)$ only one of the following conditions is true:

(1) $(e_1,\phi) \in (A,B)^v_{(e_2,\phi)}$ *(2)* $(e_1,\phi) \in (A,B)^v_{(\phi,e_2)}$

(3) $(\phi,e_1) \in (A,B)^v_{(e_2,\phi)}$ *(4)* $(\phi,e_1) \in (A,B)^v_{(e_2,\phi)}$

We can define the partial order $\leq$ on the elements of $cl(v)$ as follows.

We say $e_1 \leq e_2$, if any one of the above four conditions is satisfied.

Thus $e_1 \leq e_2$ implies that there is a diset $C \subseteq cl(v)$ such that e_1, $e_2 \in C$, and the forwardness and backwardness of e_1 and e_2 are determined by the above conditions.

Suppose $v \in P(f)$ is an extreme point of $P(f)$, as already proved in [27], a subset of v-tight constraints will form a lattice in $D(E)$ and the largest set of this lattice is sup(v) and hence will be a tight set. Since $f(o,o) = 0$, by definition $cl(v)$ is also a tight set and $sup(v) \subseteq cl(v)$.

We now show that given an orthant of E and a partial order on the elements of E, we can generate an extreme point of $P(f)$

Let (A,B) be the given orthant and let the partial order be $(e_1, e_2, ...e_n)$.

$$\text{Define } (A^0, B^0) = (\phi,\phi) \text{ and}$$

$$(A^i, B^i) = (A^{i-1} \cup e_i, B^{i-1}) \text{ if } e_i \in A$$

$$= (A^{i-1}, B^{i-1} \cup e_i) \text{ if } e_i \in B$$

(Here one has to add elements to the set (A^i, B^i) in the given partial order.)

$v \in P(f)$ is an extreme point, where i^{th} component v_{e_i} is given by the following relation.

$$v_{e_i} = f(A^i, B^i) - f(A^{i-1}, B^{i-1}) \quad if \ e_i \in A$$

$$= f(A^{i-1}, B^{i-1}) - f(A^i, B^i) \quad if \ e_i \in B$$

Theorem 4.2.2. *v generated by the above process is an extreme point of $P(f)$*

Proof. Let us first show that v is a feasible point and then we can show that it is an extreme point.

To prove that $v(C,D) \leq f(C,D)$ for every $(C,D) \in D(E)$

Notice that if $e_i \in A$ then $v_{e_i} = f(A^i, B^i) - f(A^{i-1}, B^{i-1}) \geq 0$ (since f is increasing)

and if $e_i \in B$ then $v_{e_i} = f(A^{i-1}, B^{i-1}) - f(A^i, B^i) \leq 0$

Case(1): (C,D) $\subseteq (A,B)$

In this case it is obvious that $v(C,D) \leq f(C,D)$.

Case(2):$(C,D) \not\subseteq (A,B)$

Let $F' = (e_i/e_i \in C \text{ and } e_i \in B)$ and

$F'' = (e_i/e_i \in D \text{ and } e_i \in A)$

$$\begin{aligned} v(C,D) &= v(C) - v(D) \\ &\leq v(C/F') - v(D/F'') \text{ (Since } C/F' \subseteq A \text{ and } D/F'' \subseteq B) \\ &= v(C/F', D/F'') \\ &\leq f(C/F', D/F'') \text{ (follows from Case-1)} \\ &\leq f(C,D) \quad \text{since} \quad (C/F', D/F'') \subseteq (C,D) \text{ and } f \text{ is increasing.} \end{aligned}$$

$\Rightarrow v(C,D) \leq f(C,D)$

So we have $v(C,D) \leq f(C,D) \forall (C,D) \in D(E)$

$\Rightarrow v$ is a feasible point.

Again since $|A \cup B|$=n one can easily see that there will be n tight sets corresponding to v. This proves that v is an extreme point of $P(f)$

If (A,B) is not an orthant of E then by putting $v_{e_i} = 0 \ \forall e_i \notin A \cup B$ we will obtain a $v \in P(f)$ by the above process, which may or may not be an extreme point of $P(f)$. □

In the next section, we show that the algorithm in [9] for separation problem of a polymatroid can be extended to the separation problem of an increasing bisubmodular polyhedron.

For details you may see [10].

If $M' = (E, \mathcal{O})$ is an oriented matroid with $\mathcal{O}$ as the collection of its circuits, then the unsigned circuits of $\mathcal{O}$, denoted by $\mathcal{O}'$, define a unique matroid associated with the oriented matroid $(E, \mathcal{O})$. That is

$$\mathcal{I} = \{I : C \not\subseteq I \text{ for all } C \in \mathcal{O}\}$$

is the independent sets of $(E, \mathcal{O})$, and

$$\mathcal{I}' = \{X' : X \in \mathcal{I}\},$$

are the independent sets of the matroid $(E, \mathcal{O}')$. Note that if $C \in \mathcal{O}$ then $-C \in \mathcal{O}$. If r' and r denote the rank function of $(E, \mathcal{O})$ and $(E, \mathcal{O}')$ respectively, then

$$r'(X_1, X_2) = max\{|Y_1 \cup Y_2| \ : \ (Y_1, Y_2) \subseteq (X_1, X_2), \ (Y_1, Y_2) \in \mathcal{I}\} = r(X_1 \cup X_2).$$

One can see that the rank function r' of an oriented matroid is also an increasing bisubmodular function and the corresponding polyhedron an increasing bisubmodular polyhedron.

g-polymatroids

Let g,h be any two increasing submodular functions such that,

$g(X) + h(Y) \geq g(X/Y) + h(Y/X) \ \forall X, Y \subseteq E$

Consider the g-polymatroid defined as,

$Q(g, h) = \{x \in \mathcal{R}^n \ : \ -g(A) \leq x(A) \leq h(A)\}$

Define $f(A, B) = g(A) + h(B) \ \forall A, B \subseteq E$ and $A \cap B = \phi$,

then f is an increasing bisubmodular function and $Q(g, h) = P(f)$.

4.3 General bisubmodular functions

We now extend the above concepts to a general bisubmodular function.

From now onwards f will denote a general bisubmodular function unless otherwise stated.

Let f be a bisubmodular function and $P(f)$ be its associated polyhedron.

$v \in P(f)$ be any vector in the polyhedron.

Let $\mathcal{A} = ((X, Y) |\ (X, Y)$ is $v - tight)$

As seen in the previous section if f is not increasing, then sets in $\mathcal{A}$ can be cancelling. Thus in order to define $cl(v)$, we need to talk of forward and backward elements in v-tight sets.

$cl^+(v) = (e \in E \mid\ e$ is a forward element in the v-tight sets in $\mathcal{A})$

$cl^-(v) = (e \in E \mid\ e$ is a backward element in the v-tight sets in $\mathcal{A})$

Let $cl(v) = cl^+(v) \cup cl^-(v)$

Now $\forall e \in cl^+(v)$ define

$\mathcal{A}_v^+(e) = \cap((X, Y) \mid \quad e \in X \quad (X, Y) \in \mathcal{A})$

and $\forall e \in cl^-(v)$ define

$\mathcal{A}_v^-(e) = \cap((X, Y) \mid \quad e \in Y \quad (X, Y) \in \mathcal{A})$

A graph $G(E, A)$ is said to be bidirected graph, if an edge $e \in A$ has two heads, (or) two tails (or) one head and one tail. In order to define bidirections, it is convenient to let each edge be incident to two half edges. Each half edge is incident to precisely one vertex and one edge. Let H be the set of half edges.

A biorientation (or) bidirection, over G is a function $\alpha : H \to \{-1, +1\}$.

We will denote a bidirected graph by (G, A, α). In [11] Bouchet et.el construct a jump system for a given bidirected graph as follows:

For $f' \in \mathcal{Z}^A$, and $v \in E$, the excess of f' at E is

$exf'(v) = \{\sum \alpha f'(e) \ :\ e$ is an edge incident to v$\}$

Given $c_1, c_2 \in \mathcal{Z}^A$, with $c_1 \leq c_2$, we denote by $[c_1, c_2]$ the set

$\{f' \in \mathcal{Z}^A \ :\ c_1 \leq f' \leq c_2\}$.

It has been shown in [11] that the set $\{E, ex(f') \ :\ f' \in [c_1, c_2]\}$ is a Jump system. For details see [11].

We now define the concept of a degree sequence for a bidirected graph $G(E, A, \alpha)$.

Definition 4.3.1. *Let x_v denotes the degree sequence of the bidirected graph $G(E, A, \alpha)$ then.*

x_v = (number of half edges having heads at v)-(number of half edges having tails at v).

In the above jump system put $c_1 = 0$ and $c_2 = 1$, for each edge e, then $\{ex(f'),\ f' \in [0,1]\}$ is the set of degree sequences of subgraphs of the bidirected graph $G(E, A, \alpha)$. Define

$$ex(f')(A, B) = \sum_{v \in A} exf'(v) - \sum_{v \in B} exf'(v)\ \forall (A, B) \in D(E)$$

and a function g as

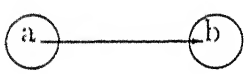
$$g(A, B) = \max_{x \in (E,\ ex(f'),\ f' \in [0,1])} (x(A) - x(B)),\ \forall (A, B) \in D(E).$$

Then $g(A, B) = exf'(A, B)$.

In what follows, given a bisubmodular function f and $P(f)$, then for any $v \in P(f)$, one can construct the bidirected graph $G(E, A, \alpha)$.

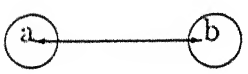
Case1:

If $(b, \phi) \in \mathcal{A}_v^+(a)$ $or(\phi, a) \in \mathcal{A}_v^-(b)$ then$(a, b) \in A$, with tail at a and head at b.



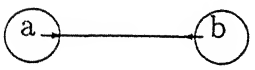
Case2:

If $(a, b) \in \mathcal{A}_v^+(a)$ $or(\phi, a) \in \mathcal{A}_v^+(b)$ then$(a, b) \in A$with heads at both a and b.



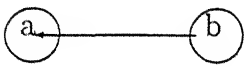
Case3:

If $(b, \phi) \in \mathcal{A}_v^-(a)$ $or(a, \phi) \in \mathcal{A}_v^-(b)$ then$(a, b) \in A$with two tails at both a and b



Case4:

If $(\phi, b) \in \mathcal{A}_v^-(a)$ $or(a, \phi) \in \mathcal{A}_v^+(b)$ then$(b, a) \in A$with tail at b and head at a.



Case5:

$\forall a \in E/cl^+(v)$ construct a self loop at a with two heads at a.

$\forall a \in E/cl^-(v)$ construct a self loop at a with two tails at a.

In what follows we talk of exchange capacities [28] of a pair of elements $\{a, b\} \in cl(v)$, with respect to $v \in P(f)$.

Exchange capacity of the pair $\{a, b\} \in cl(v)$ is a positive scalar ϵ such that an increase(decrease) of ϵ in the co-ordinate of a and a decrease(increase) of ϵ in the co-ordinate of b in the vector v still keeps the vector in $P(f)$. ϵ is the largest such positive scalar.

The following theorem from [28] gives exchange capacities in the five different cases, mentioned above.

Theorem 4.3.1. *Let f be any bisubmodular function and $P(f)$ be its associated polyhedron and $v \in P(f)$.*

1. *In Case1* $w = (v + \epsilon(a) - \epsilon(b)) \in P(f)$ *where*
$\epsilon = min\{f(X,Y) - v(X,Y) \mid (X,Y) \in D(E);\ ((a,\phi) \in (X,Y), b \notin (X \cup Y))$ *or* $((\phi,b) \in (X,Y), a \notin (X \cup Y))\}$.

2. *In case2* $w = (v + \epsilon(a) + \epsilon(b)) \in P(f)$ *where*
$\epsilon = min\{f(X,Y) - v(X,Y) \mid (X,Y) \in D(E);\ ((a,\phi) \in (X,Y), b \notin (X \cup Y))$ *or* $((b,\phi) \in (X,Y), a \notin (X \cup Y))\}$.

3. *In case3* $w = (v - \epsilon(a) - \epsilon(b)) \in P(f)$ *where*
$\epsilon = min\{f(X,Y) - v(X,Y) \mid (X,Y) \in D(E).\ ((\phi,a) \in (X,Y), b \notin (X \cup Y))$ *or* $((b,\phi) \in (X,Y), a \notin (X \cup Y))\}$.

4. *In Case4* $w = (v - \epsilon(a) + \epsilon(b)) \in P(f)$ *where*
$\epsilon = min\{f(X,Y) - v(X,Y) \mid (X,Y) \in D(E),\ ((\phi,a) \in (X,Y), b \notin (X \cup Y))$ *or* $((b,\phi) \in (X,Y), a \notin (X \cup Y))\}$.

5. *In case5*

   (a) *If* $a \notin cl^+(v)$ *then*

$w = v + \epsilon(a)$ *where* $\epsilon = min\{f(X,Y) - v(X,Y) \mid (X,Y) \in D(E); (a,\phi) \in (X,Y)\}$

   (b) *If* $a \notin cl^-(v)$ *then*

$w = v - \epsilon(a)$ *where* $\epsilon = min\{f(X,Y) - v(X,Y) \mid (X,Y) \in D(E); (\phi,a) \in (X,Y)\}$

If v is an extreme point of $P(f)$, the formulae for computing ϵ get simplified for the five cases as follows:

(i) $\epsilon = f(\mathcal{A}_v^+(a) - (b,\phi)) - v(\mathcal{A}_v^+(a) - (b,\phi))$

(ii) $\epsilon = f(\mathcal{A}_v^+(a) - (\phi,b)) - v(\mathcal{A}_v^+(a) - (\phi,b))$

(iii) $\epsilon = f(\mathcal{A}_v^-(a) - (b,\phi)) - v(\mathcal{A}_v^-(a) - (b,\phi))$

(iv) $\epsilon = f(\mathcal{A}_v^-(a) - (\phi, b)) - v(\mathcal{A}_v^-(a) - (\phi, b))$

(v) *(a)* $\epsilon = f(cl^+(v) \cup a) - f(cl^+(v))$

(b) $\epsilon = f(cl^-(v) \cup a) - f(cl^-(v))$

Since the number of co-ordinates of w differ from that of v are either one or two in all the five cases from [33], it follows that w is also an extreme point of $P(f)$. When f is an increasing function the sets $\mathcal{A}_v^{\pm}(a)$ can be calculated in $o(n^2)$ time as in the case of polymatroids [9].

We now state and prove the Min-Max theorem for a bisubmodular systems and discuss how it can be used for computing the exchange capacities ϵ defined in the previous theorem. This result has also been proved by Cunningham [16] and Fujishige [28]. Our proof differs from both.

Notation: For a given orthant (X,Y) and $y \in P(f)$

$y_{(X,Y)} \leq 0 \Rightarrow y_e \leq 0 \ \forall e \in X$ and $y_e \geq 0 \ \forall e \in Y$ We use $y \leq 0$ instead of $y_{(E)} \leq 0$. $P(f_{(X,Y)})$ is defined in chapter-3.

4.4 Min-Max theorem for a bisubmodular system

Theorem 4.4.1. *Let f be a given bisubmodular function with $f(\phi, \phi) = 0$. Let $x^o \in \mathcal{R}^E$. If there exist a $y \in P(f)$ such that $y \leq x^o$, then*

$$max\{y(E) - x^o(E) : \ y \leq x^o, \ y \in P(f)\} = min\{f(A, B) - x^o(A, B) : (A, B) \in D(E)\} \tag{4.4.1}$$

Moreover if f and x are integer valued then the maximum in the left-hand side of (4.4.1) is attained by an integral y.

Here we prove the theorem by first showing that it holds for an orthant by using the fact that $f(\phi, \phi) = 0$.

Lemma 4.4.1. *Let f be a given bisubmodular function with $f(\phi,\phi)=0$. (X,Y) be any given orthant and if there is a vector $y \in P(f_{(X,Y)})$ such that $y_{(X,Y)} \leq 0$, then*

$$max\left\{y(X,Y) \ : \ y_{(X,Y)} \leq 0; y \in P(f_{(X,Y)})\right\} = min\left\{f(A,B) \ : \ (A,B) \subseteq (X,Y)\right\} \tag{4.4.2}$$

Moreover if f is integer valued then the maximum value attained by y on the left-hand side of equation (4.4.2) is also an integer.

Proof. Define a submodular function f_1 on $E = X \cup Y$ as follows:

For any $A' \subseteq E$, let $f_1(A') = f(A_1, B_1)$ *where* $A_1 = A' \cap X$ and $B_1 = A' \cap Y$

As shown earlier in chapter-3,

$min(f(A,B)|\ (A,B) \subseteq (X,Y)) = min(f_1(A') \mid \forall\ A' \subseteq E)$

Since f_1 is a submodular function, by the min-max relationship for a submodular system in [18]

$$min(f_1(A') \mid \forall\ A' \subseteq E) = max(y(E) \mid y \leq 0, y \in P(f_1)) \tag{4.4.3}$$

If f is integer valued then f_1 is also integer valued, more over f_1 is a submodular function and hence the vector y that attains maximum at the right-hand side of the equation (4.4.3) will be an integer.

Again we have

$max(y(E) \mid y \leq 0, y \in P(f_1)) = max\left(y(X,Y) \mid y_{(X,Y)} \leq 0,\ y \in P(f_{(X,Y)})\right)$

and hence the result. □

Now we prove the theorem (4.4.1)

Proof. We prove the theorem by making use of lemma (4.4.1).

Let us first prove the simple case when $x^o = 0$. Since $x^o = 0$, by the assumption there is a $y \in P(f)$ such that $y \leq 0$. We want to prove that

$$max\left\{y(E) : \ y \leq 0,\ y \in P(f)\right\} = min\left\{f(A,B) : (A,B) \in D(E)\right\} \tag{4.4.4}$$

By the above lemma (4.4.1) for any orthant (X,Y) we have

$$max\left\{y(X,Y) \mid y_{(X,Y)} \leq 0,\ y \in P(f_{(X,Y)})\right\} = min\left\{f(A,B) \mid (A,B) \subseteq (X,Y)\right\} \tag{4.4.5}$$

Since there is a $y \in P(f)$ such that $y \leq 0$, equation (4.4.5) is valid for any orthant (X,Y) Take the min on both sides of the equation (4.4.5) over all orthants $(X,Y) \in D(E)$. The right hand side of equation (4.4.5) becomes,

$$\min_{(X,Y)} \left(min\left\{f(A,B);\ (A,B) \subseteq (X,Y)\right\}\right) = min\left\{f(A,B) :\ (A,B) \in D(E)\right\} \tag{4.4.6}$$

Similarly the left hand side of equation (4.4.5) becomes,

$$\min_{(X,Y)} \left(max\left\{y(X,Y) \mid y_{(X,Y)} \leq 0,\ y \in P(f_{(X,Y)})\right\}\right) = max\left\{y(E) \mid y \leq 0,\ y \in P(f)\right\} \tag{4.4.7}$$

Since,

$$max\left\{y(E) \mid y \leq 0,\ y \in P(f)\right\} \leq max\left\{y(X,Y) \mid y_{(X,Y)} \leq 0, y \in P(f_{(X,Y)})\right\}\ \forall \text{ orthant } (X,Y).$$

Combining (4.4.6) and (4.4.7) we obtain equation (4.4.4).

To prove the integrality of y, for any orthant (X,Y) if f is integer valued then the maximum on the left hand side of equation (4.4.5) is attained by an integral y, by lemma (4.4.1). Since we are taking the minimum over all orthants of D(E), the maximum on the left-hand side of equation (4.4.4) is also attained by an integral y.

Now we prove the min-max theorem for any $x^o \in R^E$. Consider the left hand side of equation (4.4.1).

$$max\left\{(y - x^o)(E) :\ y \leq x^o;\ y \in P(f)\right\} \tag{4.4.8}$$

Let us define a bisubmodular function g as follows:

$g(A,B) = f(A,B) - x^o(A,B)$.

$\Rightarrow \forall y \in P(f),\ (y - x^o) \in P(g)$.

Set $y' = y - x^o$ and equation (4.4.8) reduces to the following

$$max\{(y - x^o)(E) \mid y \leq x^o,\ y \in P(f)\} = max\{(y'(E) \mid y' \leq 0,\ y' \in P(g)\}$$
$$= min\{g(A, B) \mid (A, B) \in D(E)\}.$$
(From equation 4.4.4)

Replace g by f and $y' = y - x^o$, we get

$$max\{y(E) - x^o(E) \mid y \leq x^o,\ y \in P(f)\} = min\{f(A, B) - x^o(A, B) \mid (A, B) \in D(E)\}$$

Hence the theorem □

Min-Max theorem is not valid when $f(\phi, \phi) < 0$. It has been shown in chapter-2 that, when $f(\phi) < 0$, then min-max theorem in no longer valid for the submodular system.

In equation (4.4.3), we use the min-max theorem for submodular systems. Since $f_1(\phi) = f(\phi, \phi) < 0$, equation (4.4.3) may not be valid and hence the min-max relation may not hold for some orthant and hence may not also hold in general.

Computation of exchange capacities

Using the min-max relation we give methods for computing the exchange capacities, defined in theorem (4.3.1).

We give the method to calculate the value of ϵ for Case(1). The values of ϵ in other cases can be computed in a similar way.

Let $E' = E - (a, b)$. Define functions f_a *and* f_b on $D(E')$ as follows

$f_a(X, Y) = f(X \cup a,\ Y)$ and $f_b(X, Y) = f(X, Y \cup b)\ \forall (X, Y) \in D(E')$

Now define

$v_a(X, Y) = v(X \cup a,\ Y)$ and $v_b(X, Y) = v(X, Y \cup b)$

Clearly f_a and f_b are bisubmodular functions on $D(E')$ and

$min\{f_a(X, Y) - v_a(X, Y) \mid (X, Y) \in D(E')\}$

$= min\{f(X, Y) - v(X, Y) \mid a \in X$ and $b \notin (X \cup Y),\ (X, Y) \in D(E)\} = \epsilon_1$

Similarly $min\{f_b(X,Y) - v_b(X,Y) \mid (X,Y) \in D(E')\}$

$= min\{f(X,Y) - v(X,Y) \mid b \in Y \text{ and } a \notin (X \cup Y),\ (X,Y) \in D(E)\} = \epsilon_2$

$min(\epsilon_1, \epsilon_2) = \epsilon$

Since f_a, f_b are bisubmodular functions $f_a - v_a$, $f_b - v_b$ are also bisubmodular functions.

Let $h_a(X,Y) = (f_a - v_a)(X,Y)$ and $h_b(X,Y) = (f_b - v_b)(X,Y)$

In order to apply the min-max relationship the value of f on empty set must be zero. So define

$g_a(X,Y) = h_a(X,Y) - h_a(\phi,\phi)$ and $g_b(X,Y) = h_b(X,Y) - h_b(\phi,\phi)$

$min\{f_a(X,Y) - v(X,Y) \mid (X,Y) \in D(E')\} = h_a(\phi,\phi) + min\{g_a(X,Y) \mid (X,Y) \in D(E')\}$

$= h_a(\phi,\phi) + max\,(y(E') \mid y \leq 0,\ y \in P(g_a)) = \epsilon_1$

Similarly ϵ_2 can also be calculated.

From the above construction, computing exchange capacities for $v \in P(f)$ and v is not an extreme point, seems to be as difficult as that of minimising a bisubmodular function.

4.4.1 Separation problem

In this section the relationship between bisubmodular function minimisation and separation problem is given.

Definition 4.4.1. *Separation problem is to determine, given a vector x and a polyhedron $P(f)$ whether $x \in P(f)$ or not. If not, then to find a inequality of the system defining $P(f)$ that is most violated by x.*

As far as a submodular system is concerned it has been shown in [9] that the separation problem for a given x and $P(f)$ can be solved by minimizing a submodular function g defined by $g(A) = f(A) - x(A)$. Conversely the general submodular function minimization problem can be reduced to a separation problem. Now we will see this relationship in terms of a bisubmodular system. Let f be any given bisubmodular function and x the given vector. The separation problem for x and $P(f)$ is equivalent to that of minimizing

a bisubmodular function g, where $g(A,B) = f(A.B) - x(A.B)$.

If the set that minimises g is (ϕ,ϕ), then $x \in P(f)$ and if the minimum occurs at (C,D) then $x(C,D) \leq f(C,D)$ will be the most violated inequality of the system defining $P(f)$.

Let us see the converse part.

The converse problem is to find a x and a polyhedron $P(f)$ such that minimising a bisubmodular function g is equivalent to that of solving a separation problem with x and $P(f)$.

The obvious choice here is $x = 0$ and $f = g$ By the Min-Max theorem of the previous section, one can easily see that minimising a bisubmodular function g is equivalent to the separation problem for $x = 0$ and the polyhedron $P(f)$, where $f = g$.

However, is it possible to find a x and f such that f is also an increasing bisubmodular function?

For a submodular system (g,D), one can always construct a x and f increasing as in [9].

Let $x_j = g(E-j) - g(E)\ \forall j \in E$, then

$f(A) = g(A) + x(A) - g(\phi)$ is non decreasing submodular function.

Solving the separation problem for this x and $P(f)$ will also give the minimising set for g.

The validity of the above construction depends on the fact that E is the unique maximal subset in D.

But this is not the case for D(E). The number of maximal sets (orthants) in $D(E)$ is 2^n.

Thus given a bisubmodular function g it is not easy to find a $x \in \mathcal{R}^n$, such that $f = g + x$, is an increasing bisubmodular function.

However for a given orthant (X,Y) and a bisubmodular function g one can find a x, such that $f = g + x$ increasing with respect to (X,Y) as follows

Let g be a given bisubmodular function and (X,Y) a given orthant.

Let $x^*(j) = g(X-j,Y) - g(X,Y)$ if $j \in X$

$= g(X,Y-j) - g(X,Y)$ if $j \in Y$.

Define $f(A,B) = g(A,B) + x^*(A,B) - g(\phi,\phi)$, then f is increasing with respect to directed

subsets of (X,Y).

One can easily verify that minimizing the bisubmodular function g over the given orthant (X,Y) is equivalent to that of solving the separation problem with x^* and $P(f_{(X,Y)})$, where $P(f_{(X,Y)})$ is defined in chapter3.

It is possible to give a pseudo polynomial algorithm for the separation problem for an increasing bisubmodular polyhedron, by adapting the algorithm given in [9] for a polymatroid.

In the previous section we have shown that for an increasing bisubmodular function f, it is easy to obtain the sets $\mathcal{A}_v^{\pm}(e)$ for all $v \in P(f)$ We have also shown that obtaining adjacent extreme points of an extreme point in $P(f)$ will require $o(n^2)$ effort. Given that $f(\phi, \phi) = 0$, implies $x = 0 \in P(f)$ and hence $x = 0$ can be used as a starting point for the algorithm in [9].

Another pseudo polynomial algorithm for the separation problem for a polymatroid [51] requires that a polynomial time algorithm is available for solving a LPP over the given polyhedron and that adjacent extreme points can be obtained for any given extreme point of $P(f)$, for it to converge.

Since the increasing bisubmodular polyhedron satisfies both the above conditions, the algorithm in [51] can be easily adapted for solving the separation problem for this polyhedron. Integrality of f will ensure faster convergence.

If there is an oracle for minimising a bisubmodular function then by using that oracle we can minimise bisubmodular function with respect to some simple constraints that occur in some practical problems.

4.4.2 Minimizing Bisubmodular functions subjected to some simple constraints

In chapter3 we have seen that for f a bisubmodular function defined on $D(E)$, for a given diset $(X, Y) \in D(E)$

$$\min_{(A,B)\subseteq(X,Y)} f(A, B)$$

is equivalent to

$\min_{A\subseteq\bar{E}}(\bar{f}(A))$ where $\bar{E} = (X \cup Y)$ and $\bar{f}$ is a submodular function defined as follows

$$\bar{f}(A) = f(A_1, A_2), \quad \text{where } A_1 = (A \cap X) \text{ and } A_2 = (A \cap Y).$$

In the lemma below we detail a scheme for finding a nonempty minimising diset for a bisubmodular function. We state the problem as:

$$\min_{(\phi,\phi)\neq(A,B)\in D(E)} f(A, B) \tag{4.4.9}$$

Lemma 4.4.2. *Problem (4.4.9) can be solved by using an oracle for minimising a bisubmodular function not more than $2|E|$ times.*

Proof. For $j \in E$,set $E^j = E - j$.

Define

$f_{(j,\phi)}(A, B) = f(A \cup j, B)\ \forall (A, B) \in D(E^j)$ and

$f_{(\phi,j)}(A, B) = f(A, B \cup j)\ \forall (A, B) \in D(E^j)$

Clearly the above two functions are bisubmodular on $D(E^j)$.

Now using the oracle for minimising a bisubmodular function find

$$\min_{(A,B)\in D(E')}(f_{(j,\phi)}(A, B))$$

$$\min_{(A,B)\in D(E')}(f_{(\phi,j)}(A, B))$$

Similarly for all other $j \in E$ find the minimising disets for the corresponding bisubmodular functions. Among all the minimising sets find the one which corresponds to the minimum value. The set we obtain is the minimum nonempty set that minimizes f. □

Consider the following constrained bisubmodular function minimisation problem

For a given $i \in E$,

$$\min_{(i,\phi) \not\subseteq (A,B) \in D(E)} f(A,B)$$

Lemma 4.4.3. *The above problem can be solved by using the oracle for minimising a bisubmodular function not more than 2 times.*

Proof. Let $E^i = E \backslash i$

$$\min_{(A,B) \subseteq D(E^i)} f(A,B) = f(A_1,B_1), \text{say}$$

Now define $f_{(\phi,i)}(A,B) = f(A, B \cup i)$. Clearly $f_{(\phi,i)}$ is a bisubmodular function on $D(E^i)$.

$min_{(A,B) \subseteq D(E')} \left(f_{(\phi,i)}(A,B)\right). = f(A_2,B_2)$.

Let $f(A,B) = min\{f(A_1,B_1)\ ,\ f(A_2,B_2)\}$

(A,B) is the required minimum set. □

Given a diset $(X,Y) \in D(E)$, such that $|X \cup Y| \geq 2$, we now want to find a minimising set (A,B) that has atleast one element in common with (X,Y) and no more than $(|X \cup Y| - 1)$ elements. The problem can be stated as follows:

$$\min_{(\phi,\phi) \subset (A,B) \in D(E)} (f(A,B) : 1 \leq \mid (A,B) \cap (X,Y) \mid \leq \mid X \cup Y \mid -1)$$

Lemma 4.4.4. *The above problem can be solved by using an oracle for minimising a bisubmodular function atmost $|E^2|$ times.*

Proof. Let (X,Y) be the given nonempty diset.

Let $(i,\phi) \subseteq (X,Y)$. Note that here i can also be a backward element of (X,Y). With out loss of generality we will assume that i is a forward element.

Define $f_{(i,\phi)}(A,B) = f(A \cup i, B) \quad \forall (A,B) \in D(E \backslash i)$

Since $\mid X \cup Y \mid \geq 2$,let $(j,\phi) \subseteq (X,Y)$

Here also j can be backward element of (X,Y).

Now find

$$\min_{(\phi,\phi)\neq(A,B)\in D(E\setminus i)} \left(f_{(i,o)}(A,B) \quad (j,\phi) \notin (A,B)\right)$$

One can find the above minimum by using lemmas (4.4.3) and (4.4.4).

Now repeat the above steps by fixing one element of (X,Y) and varying the others.Find the minimum among all the sets. So the complexity of the algorithm is $o(|E|^2)$. □

There are practical situations where we want to minimise a bisubmodular function with some simple constraints as above.

In many problems, the diset (ϕ,ϕ) will be a trivial solution for minimising a bisubmodular functions. However we need only non trivial solutions. In such situations lemma (4.4.3) will be useful. If (X,Y) is a given diset with $|X \cup Y|$=even, then one is required to find a minimising set (A,B) of f such that $|(A,B) \cap (X,Y)|$=odd. Lemma (4.4.5) can be used to solve the above problem. These type of problems occur while solving the separation problem over $P(f)$ defined by odd set constraints with respect to (X,Y). It has been proved in the previous section that the separation problem is equivalent to that of minimising a bisubmodular function.

Chapter 5

Minimum cost Network and Submodular flow problems

5.1 Introduction

The minimum cost network flow problem is one of the fundamentals of all network flow problems. The cycle-cancelling algorithm was one of the first algorithms developed to solve this problem. This algorithm maintains a feasible solution at every iteration. It proceeds by identifying negative cycles in the residual network and augments along these cycles until there are no negative cycles in the residual network. The cycle cancelling algorithm was suggested by Klein [39] and the generic version of this algorithm is known to run in pseudo-polynomial time. Goldberg and Tarjan [31] developed a strongly polynomial algorithm which cancels minimum-mean cycles. To obtain a better bound, they used the idea of indirect cost scaling with cycle-cancelling; cycles composed entirely of negative reduced cost arcs are cancelled and optimality violation of arcs are tightened by updating node potentials. The other cycle-cancelling algorithm by Barahona and Tardos [7] augments flow along all negative cycles with maximum improvement. Sokkalingam [52] developed an algorithm which augments flow along a negative cycle with sufficiently large residual capacity.

Submodular flow problems are the generalization of classical network flow problems. They

are the network flow problem with boundary constraints described by the submodular functions. A straightforward algorithm for solving submodular flow problem is the cycle cancelling algorithm.

In this chapter we give a new cycle cancelling algorithm for the minimum cost network flow problem and for solving the submodular flow problem.

Our cycle cancelling algorithm for minimum cost network flow problem works as follows: We propose a search procedure which looks for augmenting cycles as well as dynamically updates potential of the node blocking the search. The basic idea is to reduce optimality violation of all arcs incident to the node blocking the search by half. The important feature of our algorithm is that it simultaneously finds negative cycles and tightens optimality violations by using the basic graph search algorithm.

We extend the algorithm to solve the minimum cost submodular flow problems.

The organisation of this chapter is as follows:

We first review some relevant definitions and results. Next we propose a new version of cycle cancelling algorithm. Following this we deals with the complexity analysis. We then study the computational experimentation of our algorithm. Here we compare our algorithm experimentally with the relaxation algorithm of [8] Computational results are also given. The c-code for our algorithm is given in the Appendix. At the end we propose a new cycle cancelling algorithm for the minimum cost submodular flow problem.

5.2 Preliminaries

In this section, we present network notations and review some well known results used in the subsequent sections.

Problem Statement

Let $G = (V, A)$ be a directed network defined by a set V of n nodes and a set A of m arcs. Each arc (i, j) has an associated cost c_{ij}, and an associated capacity u_{ij}. We denote by C the largest magnitude of all arc costs. We associate with each node $i \in V$ a number $b(i)$ representing its supply/demand such that $\sum_{i \in N} b(i) = 0$. The minimum cost flow problem is to determine the flow x_{ij} on each arc $(i, j) \in A$ which solves the following decision problem:

$$\text{Minimize} \sum_{(i,j) \in A} c_{ij} x_{ij} \tag{5.2.1}$$

subject to

$$\sum_{\{j : (i,j) \in A\}} x_{ij} - \sum_{\{j : (j,i) \in A\}} x_{ij} = b(i) \quad \text{for all } i \in V, \tag{5.2.2}$$

$$0 \leq x_{ij} \leq u_{ij} \ \forall \ (i, j) \in A. \tag{5.2.3}$$

We refer to a flow x satisfying the mass balance constraints (5.2.2), and the flow bound constraints (5.2.3) as a **feasible flow.** Our algorithm maintains a feasible flow at every step and proceeds by improving its objective function value. Without any loss of generality, we assume that the minimum cost flow problem ((5.2.1) – (5.2.3)) is feasible and its optimal objective function value is finite. Verifying both the conditions will not require much effort. Existence of a feasible flow can be answered by solving a max-flow problem and finiteness of the objective function is guaranteed by the absence of uncapacitated negative cost cycles. This can also be verified in o(nm) time.

Residual Network $G(x)$

The residual network $G(x)$ corresponding to a feasible flow x is defined as follows: For each arc $(i, j) \in A$ with $x_{ij} < u_{ij}$ replace (i, j) by an arc (i, j) with cost c_{ij} and residual

capacity $r_{ij} = u_{ij} - x_{ij}$. For each arc (i,j) with $x_{ij} > 0$ replace (i,j) by an arc (j,i) with cost $c_{ji} = -c_{ij}$ and a residual capacity $r_{ji} = x_{ij}$. The residual network consists only of arcs with positive residual capacity.

For a given set π of node potentials, the reduced cost of an arc (i,j) in the residual network is $c_{ij}^{\pi} = c_{ij} - \pi(i) + \pi(j)$. An arc (i,j) $\in$ G(x) is said to be eligible if $c_{ij}^{\pi} < 0$. Our algorithm often works with reduced costs in place of arc costs and makes use of the following well-known property:

Property 1. For any directed cycle W in G(x) and any set of node potentials π $\Sigma_{(i,j)\in W} c_{ij}^{\pi} = \Sigma_{(i,j)\in W} c_{ij}$.

Minimum Cost Optimality Conditions

Optimal solutions of the minimum cost flow problem can be characterized in a variety of ways. The theorems below give these characterizations and some related results. For details see Ahuja et al.[3].

Theorem 5.2.1. Negative Cycle Optimality Conditions. *A feasible solution x is an optimal solution of the minimum cost flow problem if and only if the residual network $G(x)$ contains no negative cost directed cycles.*

Theorem 5.2.2. Reduced cost optimality conditions. *A feasible solution x is an optimal solution of the minimum cost flow problem if and only if some potential π satisfies the following reduced cost optimality conditions:*
$c_{ij}^{\pi} \geq 0$ for every arc (i,j) in $G(x)$

Theorem 5.2.3. ϵ—optimality conditions: *A flow x is said to be ϵ optimal for some $\epsilon > 0$ if x together with some node potentials π satisfies the following optimality conditions:*
$c_{ij}^{\pi} \geq -\epsilon$ for every arc (i,j) in $G(x)$.

We will also denote ϵ as a measure of **violation of optimality**.

Theorem 5.2.4. *For a minimum cost flow problem with integer costs, any ϵ-optimal feasible flow is an optimal flow if $\epsilon < 1/n$.*

Theorem 5.2.5. *Suppose that x is an optimal flow with respect to the potentials π, and suppose that for some arc $(k,l) \in A$, $|c^{\pi}_{kl}| \geq 2n\epsilon$. Then the flow on arc (k,l) remains the same for all ϵ'-optimal flows for $\epsilon' \leq \epsilon$.*

5.3 A New Cycle cancelling Algorithm

Reducing Optimality Violation by Dynamic Sharing

In our algorithm, the main procedure searches for a negative cycle and when the search is blocked at a node, dynamically shares the optimality violation at the blocked node, if it is sufficiently large. Let ε_0 denote the current maximum violation. The aim at each iteration is to reduce the optimality violation of arcs incident to nodes visited by the search procedure to below $\varepsilon_0/2$ at its termination. We call this procedure as **dynamic-share**, which is described in details below.

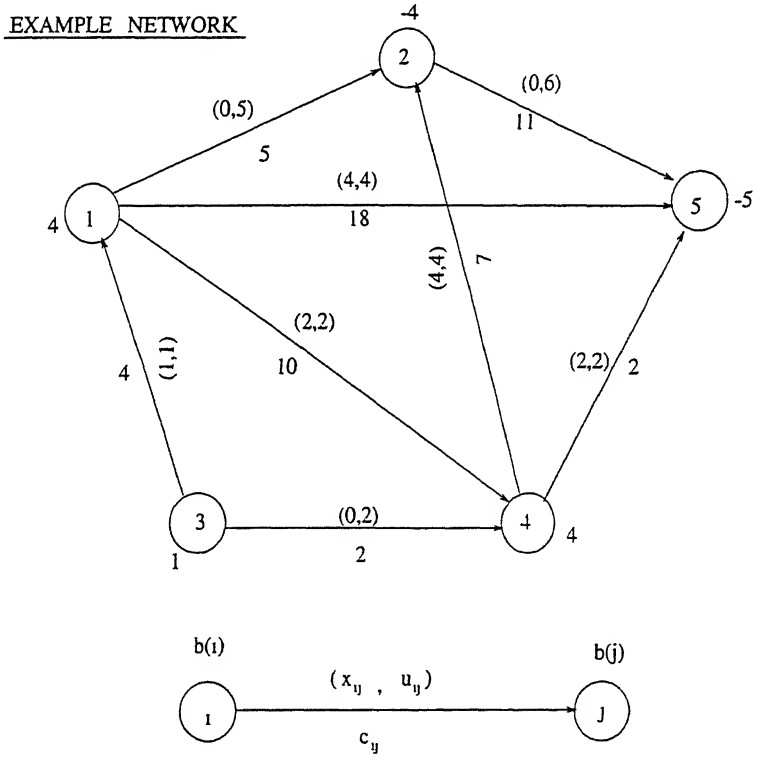
To find a negative cycle, we start building a path by successively adding an eligible arc to the tip of the path. For that,we choose a node with backward eligible arcs, whose maximum violation is greater than $\varepsilon_0/2$, as the origin of a path. Initialy, the path contains no arcs and the tip of the origin is same as the the origin of the path. When we retreat from the origin, the path becomes null.

We continue adding an eligible arc to the tip of the path until either of the following conditions results: 1) adding an arc forms a cycle, 2) there is no forward eligible arc at the tip. In the former case, we improve the solution by augmenting flow along the cycle, we start the procedure again at the first node common to both the path and the cycle.

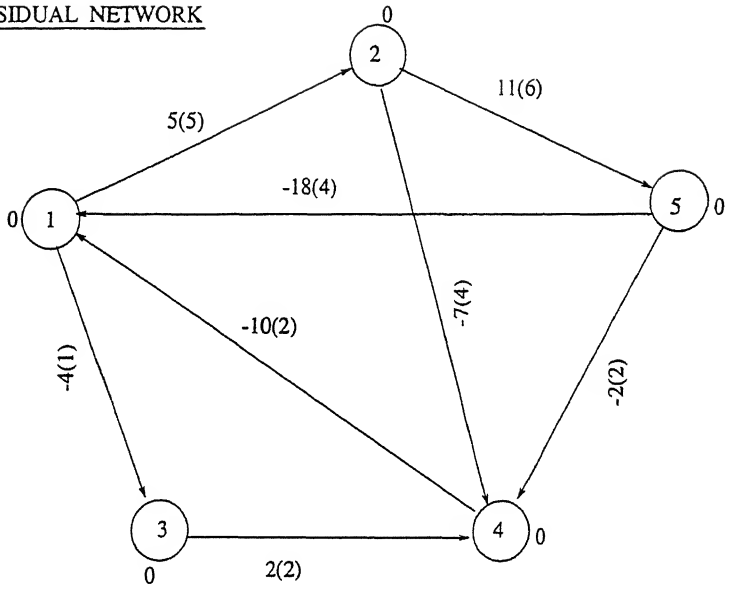
In the later case, the search is said to be blocked at the tip. We calculate the maximum violation, say α, of backward arcs incident on the tip. If $\alpha > \varepsilon_0/2$, we share, that is we increase the potential of the tip only by $\alpha/2$; note that always $\alpha < \varepsilon_0$. Hence, after the

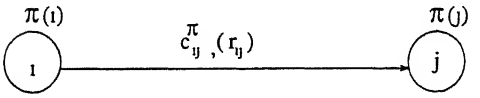
update, optimality violation is shared equally between forward and backward arcs. We refer to this update as **sharing update**. Otherwise, we transfer, i.e., we increase the potential of the tip by α. After this update, the whole optimality violation of backward arcs are transfered to forward arcs. We refer to this update as **transfer update**. After potential updates, we retreats to the predecessor of the tip on the path. We terminate the procedure when we have retreated to the origin.

In what follows is an example network and its corresponding residual network and an example of sharing update and transfer update.



RESIDUAL NETWORK

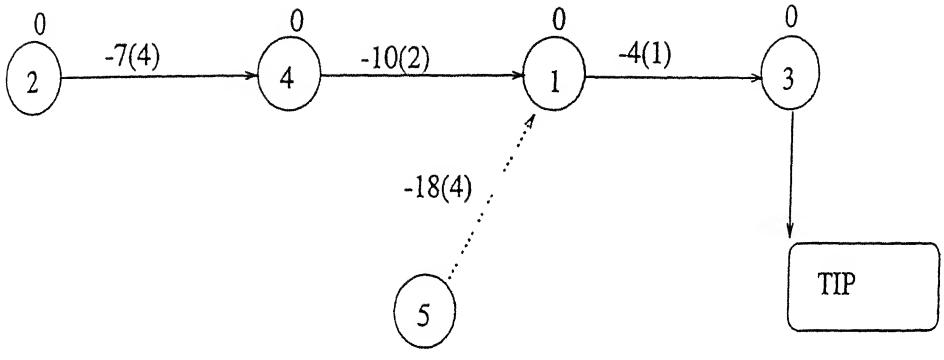




Now max $\{|c_{ij}|\} = \epsilon_o = 18$ and $\epsilon_o/2 = 9$

At some stage of the iteration we have the path as given below with c^{π}_{ij} and r_{ij} shown along the arcs. Now the path is blocked and the tip node is 3.

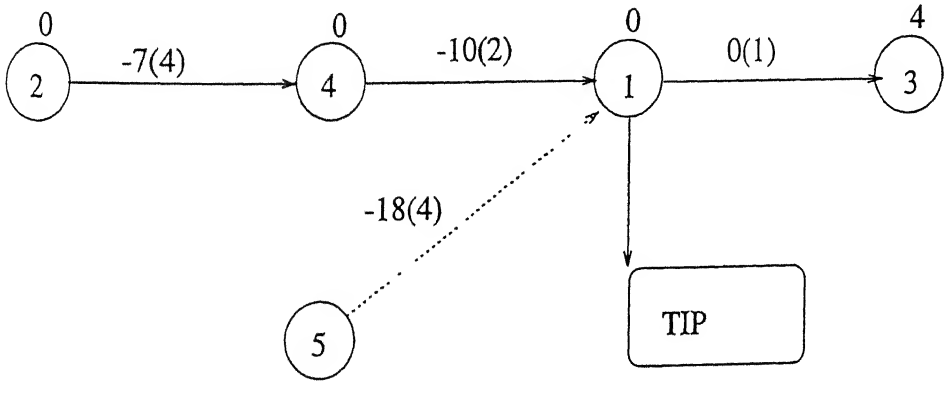
TRANSFER UPDATE



That is no forward eligible arc from node 3.

Here α = max violation of all backward arcs at node 3. Here the only backward arc at 3 is the arc(1,3) which has optimality violation 4.

So $\alpha = 4 < 9$ and hence we have a transfer update Set $\pi(3) = \pi(3) + \alpha = 4$ and the tip to the predecessor of (3). After the update the path is,



Now one cannot move from node 1, as there is no forward eligible arc. Calculate the

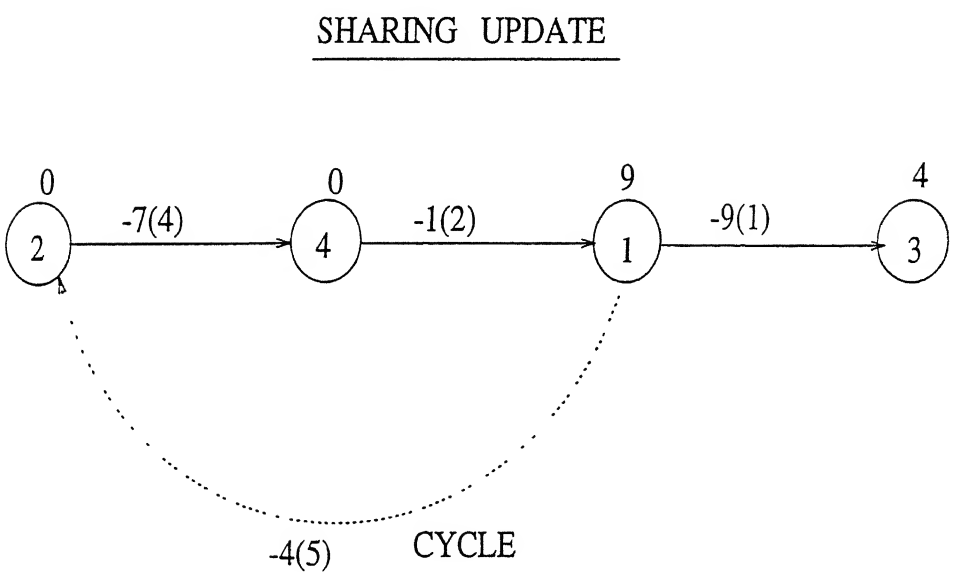
maximum violation of all backward arcs at 1. Here (4,1) and (5,1) are two backward arcs at node 1,

$\alpha = max(10, 18) = 18 > \epsilon_o/2 = 9$,

and hence we have to update the potential of the tip node by sharing update.

So $\pi(1) = \pi(1) + \alpha/2 = 9$

After this update you can either move from 1 to 2 or from 1 to 3. If you choose the path



from 1 to 2 then there is a cycle and hence augmentation takes place.

Algorithm: Dynamic Share

The algorithmic description is given below. Let r refer to the origin, Path refer to the path built by the algorithm from the origin to the tip, and t to the tip of Path.

```
Procedure Dynamic share(r, ε0)
begin
                    set r as the root of Path;
                     while Path is not null do
          begin
                     find an eligible forward arc of t, if it exists;
                     if successful then
            begin
                     add the arc to t;
                     if cycle is formed then
                  begin
                     augment flow along the cycle formed;
                     set the tip t to the first node common to both path and cycle;
                  end;
                       else set t to the head of the new arc;
            end;
                 else
                   begin
                     compute the maximum violation of backward arcs;
                     if it is greater than the critical limit then share;
                     else transfer;
                     set the tip to the predecessor of t;
                   end;
          end;
end;
```

We now prove some properties of the procedure **dynamic_share.** Let bmin(t) denote the minimum of reduced costs of backward arcs incident on t, and bviol(t) its absolute value. The following proposition is straight forward.

Proposition 5.3.1. *Increasing the potential of a node by δ units (i) decreases reduced cost of each forward arc of the node by δ units, and increases reduced cost of each backward arc by δ units.*

Lemma 5.3.1. *After a* **transfer-update** *at a node, (i) there will be no backward eligible arcs. (ii)all forward arcs will have optimality violation $\leq \epsilon_o/2$.*

Proof. (i) Let us assume that the transfer update took place at a node t. The backward arcs at t are $(i_1, i_2..., i_r)$ say. Let ϵ_o denote the current maximum violation.
Since it is a transfer update $bviol(t) \leq \epsilon_o/2$ and $-bviol(t) \leq c^{\pi}_{i_k t}\ \forall\ k = 1...r$
After the update $\pi'(t) = \pi(t) + bviol(t)$
The reduced cost of all backward arcs after the update are as follows:
$c^{\pi}_{i_k t} = c_{i_k t} - \pi(i_k) + \pi'(t) = c_{i_k t} - \pi(i_k) + \pi(t) + bviol(t) \geq 0\ \forall\ K = 1\ ..\ r.$
Which implies that all the backward arcs are ineligible
(ii) Before the transfer update, there were no eligible forward arcs at t, that is the optimality violations are zero for all the forward arcs at t. We decrease the reduced cost of all the forward arcs by a quantity $bviol(t) \leq \epsilon_o/2$ and hence the optimality violation of all the forward arcs are $\leq\ \epsilon_o/2$ □

Lemma 5.3.2. *After a sharing-update at a node, each forward and backward arc will have optimality violation less than $\epsilon_o/2$, where ϵ_0 is the current optimality violation of the flow.*

Proof. Let us assume that the sharing update took place at a node t. So $bviol(t) > \epsilon_o/2$. (Note that $bviol(t) < \epsilon_o$)
After the update $\pi'(t) = \pi(t) + bviol(t)/2$

By the above proposition (5.3.1) reduced cost of all forward arcs will get decreased by the amount $bviol(t)/2$ and for backward arcs will get increased by $bviol(t)/2$. Before the sharing update, there were no forward eligible arcs, that is optimality violations of all the forward arcs at t were zero and hence the optimality violations of all the forward arcs at t $\leq \epsilon_o/2$

As far as the backward arcs at t are concerned, it is easy to see that increase in the reduced cost by the amount $bviol(t)/2$ makes their optimality violations $\leq \epsilon_o/2$. □

Observe that no eligible arcs are created during flow augmentation, because after augmenting flow along a cycle, only the reverse of the arcs on the cycle get added to the residual graph and by definition these arcs will have positive reduced cost and hence will not be eligible. Arcs in the cycle saturated by flow augmentation drop out of the residual network. Operations augmentation and update neither introduce new eligible arcs nor change the reduced costs. With these observations, we now prove the following theorem.

Theorem 5.3.1. *Let $\epsilon = \epsilon_0/2$, where ϵ_0 is the optimality violation of the flow, denote the* **target-approximation**. *Procedure dynamic-share has the following properties. (i) Between two consecutive sharing-updates, any node is (transfer) updated at most once. (ii)At any node, there will be at most one sharing-update. (iii) At the termination, arcs incident to nodes visited by the procedure have optimality violation less than or equal to the target-approximation.*

Proof. (i) Let a transfer update take place at tip t. By lemma 5.3.1 (i), there are no backward eligible arcs at t, and since after the update the arc (predecessor(t),t) has become ineligible, we begin the search from predecessor(t). In order to reach node t, a sharing update must take place at one of the tails of an arc incident on t. Hence another transfer update at t can take place only after a sharing update has taken place.

Thus we have proved statement (i).

(ii)A sharing update occurs at a node, say t, only when bviol(t) $> \epsilon$. By Lemma (5.3.2) , after the update, bviol(t) is $\leq \epsilon$. Also, by Lemma (5 3.1) and Lemma (5.3.2) together with the observation in the para preceding theorem(5.3.1), optimality violation of new eligible arcs is $\leq \epsilon$. Hence bviol(t) continues to be $\leq \epsilon$ after sharing-update. Hence there will be no more sharing-updates.

(iii) Optimality violation of all forward arcs of a node is $\leq \epsilon$ after an update at that node. Optimality violation of forward arcs may only decrease due to augmentation and updates at other nodes. Optimality violation of backward arcs is $\leq \epsilon$ after any update (it is zero after transfer updates). Optimality violation of backward arcs may increase due to updates at other nodes. As we pointed out in the previous para, optimality violation of new eligible arcs is $\leq \epsilon$. Hence optimality violation of backward arcs of any node visited remains $\leq \epsilon$. Hence statement (iii) follows. □

5.3.1 Current Arc Implementation.

We implement the algorithm using the **forward-backward-star** data structure. This enables us to update and calculate the reduced cost on any arc in $O(1)$ time as well scan forward star (backward star) of any node i in $O(|A(i)|)$ ($O(|B(i)|)$) time. Where A(i)=set of all forward arcs from node i and B(i)= set of all backward arcs from node i.

Further, to reduce the time for finding eligible arcs at the tip of the path, we use the **current-arc** data structure on top of the forward-star data structure. For each node i, we define a **current arc** (i, j), as the arc we wish to examine for eligibility when node i is visited. With this data structure we can do following operations in $O(1)$ time.

set(i). =(i,j), where j is the first node for which (i,j) $\in$ forward star of i.

get-arc(i). Return the current arc of node i.

move-arc(i). If the current arc is not the last arc of forward star of node i, set the current arc to the next arc in the forward star; otherwise, return end of list.

To use the current-arc data structure in the procedure dynamic-share, we set the current arc of each node to its first forward arc; when a node becomes the tip by get-arc operation, we get the current arc of that node, and test whether the arc is eligible and has non-zero residual capacity . If the answer is yes then, current arc of the node is added to the path; otherwise, the operation move-arc(i) is used and residual forward arc is found. If no such arc can be found, end of the forward star will be reached and the node will be updated and the current arc will be reset to the first arc of the forward star of the tip node. The path will retreat to the predecessor of the node.

We will verify the correctness of the implementation. Let i be any node. It is enough to prove that there is no eligible forward arc, at node i, when the current arc reaches the end of the forward star of node i. This can be seen as follows:
The current arc moves to the next arc in the forward star of node i only when it becomes ineligible or its reduced capacity becomes zero. (ie. the arc drops out of the residual graph). Hence all the arcs of the forward star of node i scanned before the current arc remain ineligible and (or with zero residual capacity). So at the end, all the arcs in the forward star of node i will become ineligible.

Successive Dynamic Sharing algorithm

In this section, we describe the procedure which finds minimum cost flow by applying ıteratively the procedure dynamic share descrıbed ın the previous section. For the moment, we assume that costs are integral.

The algorithm starts with a feasible flow and a set of node potentials π. Initialy, we can start with $\pi = 0$. Each major iteration consists of the following steps. We first find the maximum optimality violation over all arcs, say θ If $\theta < 1/n$, we have an optimal flow and we terminate the algorithm. Otherwise we set the **target approximation,** say ϵ, for procedure share to be applied in this iteration as $\theta/2$. We select a node, say r, with

eligible backward arcs whose maximum optimality violation is greater than ϵ. We call such a node as candidate node. We apply the procedure dynamic_share setting node r as the root node. The next iteration starts when there are no candidate nodes.

5.4 Complexity Analysis

We now analyze the complexity of the algorithm. Any arc (i, j) may become an eligible arc again during the procedure when the potential of its tail i increases. Note that flow augmentation does not create any eligible arcs. Since the potential of any node increases at most n times during the procedure, any arc can becomes eligible after being ineligible atmost n times . Since augmentation removes at least one arc from the residual graph by saturating it, this observation implies that there can be at most $O(nm)$ flow augmentations. Since each augmentation takes $O(n)$ time. the time taken by the flow augmentation is $O(n^2m)$.

We now account for the time taken by arc scanning. Between two consecutive scanning each ineligible arc is scanned at most once. But, an eligible arc may be scanned when path retreats either (i) after augmentation or (ii) after sharing-update. We can include the time taken by arc scanning in (i) to the flow augmentation time, and the time due to (ii) to the number of sharing updates which is $0(n)$. Since the number of updates at a node is $0(n)$, the time for arc scanning is $O(\Sigma_{i \in N} |A(i)|) = O(nm)$. The time taken by the retreat operation is equal to the number of updates, which is $O(n^2)$. The time taken by operations add-arc can be charged to the operations flow augmentation and retreat. Thus overall complexity of the procedure is $O(n^2m)$. We now state the above observations in the following lemmas.

Lemma 5.4.1. *Using the* **current-arc** *data structure, procedure dynamic-share can be implemented in* $O(n^2m)$ *time.*

Lemma 5.4.2. *Each major iteration of the algorithm successive dynamic sharing reduces the maximum optimality violation by a factor of 2 and runs in $O(n^2m)$ time using the current-arc data structure*

The maximum optimality violation ϵ of any feasible flow is at most C at the beginning of the algorithm. Since each major iteration reduces optimality violation by a factor of 2, after $O(\log(nC))$ iterations optimality violation will reduce to $1/n$. By Theorem (5.2.4), when costs are integral, flow obtained after $O(\log(nC))$ iterations will be optimal.

As with other cost scaling algorithms, this algorithm can be modified for real data costs using the following well-known technique. We divide major iterations into a group of $\log\lceil n\rceil + 1$ iterations, and each iteration starts with solving a minimum mean cycle problem. Let λ denote the minimum mean value. By adding λ to each arc in the residual graph and solving a shortest path problem, we can update potentials such that $\epsilon = \lambda$. In particular, each arc in the minimum mean cycle has its reduced cost equal to $-\lambda$. After $\log\lceil n\rceil + 1$ iterations in the group, $\epsilon \leq \lambda/2n$; this implies at least one arc in the minimum mean cycle found at the start of this group of iteration will have reduced cost $\geq 2n\epsilon$. Fix the flow on that arc. Details can be found in Goldberg et. al.[31] and Ahuja et. al [3]. This implies we will get the optimal flow in m group of major iterations, each fixing flow on one arc. Thus, incorporating the minimum mean cycle routine as above, the algorithm solves the minimum cost flow problem in $O(m\log n)$ major iterations.

Theorem 5.4.1. *Successive Dynamic Share Algorithm performs $O(\log(nC))$ iterations for integral data and runs in $O(n^2m\log(nC))$ time. By incorporating minimum mean cycle routine into the algorithm, for real data, the algorithm performs $O(m\log n)$ iterations and runs in $O(n^2m^2\log n)$ time.*

5.5 Computational Experimentation

5.5.1 Introduction

Historically, computational experimentation has been the primary method for comparative evaluation of network flow algorithms. During the 1960's it was generally believed that primal-dual methods hold advantage over primal simplex methods.

However in 1970's use of sophisticated data structures propelled primal simplex algorithms to a position of prominence for solving general minimum cost flow problems.The different primal-dual algorithms in use are the out-of-kilter method, Relaxation algorithm and the successive shortest path algorithm. Almost all the primal methods are variations of the cycle-cancelling algorithm.

Bertsekas and Tseng [8] compared their relaxation code(RELAX-II) with two state-of the-art codes:KILTER (a primal-dual code due to Aashtiani and Magnanti) [1] and RNET (a primal simplex code due to Grigoria and Hsu 1980). Among the available codes at that time it appears that RNET had produced the fastest computations for the NETGEN [40] bench mark problems.

But the results of (Bertsekas) [8] show the substantial superiority of the RELAX-II code over the other codes. For the Assignment and Transportation problems computational time of RELAX-II code is almost five times better than KILTER and RNET, and for uncapacitated and lightly capacitate problems it out performs the other codes. The margin of superiority increases for large problems.

In this section we compare our algorithm, described in the previous section with the RELAX-II code.

The test problems used are the standard bench marks generated by the publicly available NETGEN program [40]. We call our code as CYCLE-code.

5.5.2 Computational Implementation

As far as the implementation is concerned, in each iteration during updating potential at tip nodes, division by a factor of 2 occurs thus making the value of ϵ non integral. However by slight modification of the costs we can ensure that ϵ remains integral. We do so by multiplying arc costs by number of nodes n and setting the initial value of $\epsilon = 2^{log(nC)}$. We stop the algorithm when $\epsilon < 1$.

But while doing so, the values of costs become very large, thus making the computational efficiency less. So to avoid this we do the following·

Start the algorithm with the given costs, each time while calculating $\epsilon/2$ take its integral part and stop the algorithm when $\epsilon < 1$. Now the flow is ϵ-optimal but not optimal. To make it optimal we apply the shortest-path algorithm as follows.

For each node calculate its demand(supply) as follows Let b(i) denote the supply(demand) for the node(i) Set b(i)=0 for all the nodes. For every admissible arc (i,j) do the following:

begin

if$\Big((x_{ij} < u_{ij})$ and $(c^{\pi}_{ij} = -1)\Big)$

begin

b(i)=$b(i) - (u_{ij} - x_{ij})$ and b(j)=b(j)+$(u_{ij} - x_{ij})$

x_{ij}=u_{ij}

end

if$\Big((x_{ij} > 0)$ and $(c^{\pi}_{ij} = 1)\Big)$

begin

b(i)=b(i)+x_{ij} and $b(j) = b(j) - x_{ij}$

x_{ij}=0

end

end

After the above steps. nodes with $b(i) > 0$ are supply nodes and with $b(i) < 0$ are demand ones. Now select a supply node i and a demand node j and find the shortest path from node i to j, with reduced cost c_{ij}^{π} as the distance from node i to j.

After finding the shortest path augment the flow along the path and also calculate the distance of the shortest path from i to j, for updating the node potentials. Change the supply and demand for nodes i and j respectively and proceed. These steps are explained below.

Shortest-path(i,j) is a procedure to find the shortest-path from i to j.

Repeat

begin

(Let i be the supply and j be the demand node)

CALL shortest-path(i,j).

Augment the flow along the path.

Let delta be the amount of flow augmented.

b(i)=b(i)-delta;

b(j)=b(j)+delta;

For all the nodes in the shortest path change the potentials as follows.

$\pi_j = \pi_j - c_{ij}^{\pi}$ +distance of the shortest path.

end

until(demand and supply for all the nodes are satisfied.)

Now the flow is optimal.

5.5.3 TEST CONDITIONS

All problems were tested under identical conditions. We worked on HP-9000/800 mainframe series. We use the language C. The RELAX-II code available to us was in Fortran-iv. We used FORTRAN to C converter and convert the Relax-II code to C language.

We obtain the CPU times using system command TIME in Unix. The time command will

give three times, real,user,system. The CPU time is user time plus system time.

To obtain a starting feasible flow for our algorithm we use a maximum flow algorithm. The computational investigations of maximum flow algorithm by R.K Ahuja et.el [2] conclude that the networks generated by NETGEN are rather easy classes of networks for maximum flow algorithms. In most of the example problems we took in this section, we found that the time taken for solving the max flow is negligible. So in our study, the time taken for solving the maximum flow problem is not included.

5.5.4 Computational Results

We have organised our computational results for the Cycle-code into five tables. All the problems shown were generated using the widely used, publicly available NETGEN program. We used the same random seed number 13502460 (the number used by the authors of NETGEN). We also provide all additional information needed to replicate these problems.

Table-I (Standard NETGEN Bench Marks)

This table shows the result for Cycle-code and Relax-II code for the problems described in detail by Klingman, Napier and Stutz [40], generated by the NETGEN program. The results shows that for the transportation problems and uncapacitated and lightly capacitated problems, both Relax-II code and Cycle-code have very close performance with Relax-II code having a slight upper edge over our Cycle-code

As it has already been reported in [8] Relax-II code performs much better than RNET and KILTER for these problems, we can say that Cycle-code has an edge over RNET and KILTER.

For the assignment problems interestingly, Cycle-code has a slight edge over Relax-II code.But for the uncapacitated and lightly capacitated problems with large nodes clearly Relax-II code is substantially faster than the cycle-code

Table-2 (Assignment problems with large cost range)

These results are in general agreement with those for the assignment problems of Table-I. Here the cost range is from 1-10000. For future reference, we also give the optimal values of the objective functions.

Table-3 (Lightly capacitated and high density problems)

The problems here have a large number of arcs compared to the number of nodes. One can see from the table that the increase in the number of arcs does not affect the computational behaviour of Cycle-code and its performance is again very close to that of the Relax-II code.

Table-4 (Problems with lesser flow values)

In this table we study the nature of problems which have less flow values but are dense. From the data available from this table one can see that here too. the performance of the Cycle-code is comparable to that of the Relax-II code.

Table-5 (Transportation problems with large cost range)

The results in the table are in general agreement with those of the problems 1-10 in Table-1. Both the Cycle-code and the Relax-code are performing very close. Even with large costs range from 1-10000 the performance of the Cycle-code, is not affected.

5.5.5 Conclusion

Based on the computational results in Table 1-5, we can conclude that our Cycle-code is comparable to Relax-II code for both standard bench mark problems and a broad range of randomly generated problems using NETGEN. As mentioned earlier, Berstsekas and Tseng [8] have shown that their Relax code is much faster than all the classical methods. The advantage of our algorithm is in its implementation, which is very simple. The performance is also very impressive. The Relax-II method adapt non linear programming concepts to solve the linear network flow problems.

Table 5.1: Standard Benchmark problems of Klingman. Napier and Stutz, obtain using NETGEN

Problem type	No.	No.of Nodes	No.of Arcs	CYCLE-code	Relax-code
Transportation	1	200	1300	0.14	0.13
	2	200	1500	0.19	0.16
	3	200	2000	0.29	0.20
	4	200	2200	0.32	0.21
	5	200	2900	0.28	0.26
	6	300	3150	0.41	0.30
	7	300	4500	0.68	0.40
	8	300	5155	0.60	0.45
	9	300	6075	0.91	0.51
	10	300	6300	0.85	0.60
Total Time(1-10)				4.67	3.22
Assignment	11	400	1500	0 12	0.17
	12	400	2250	0.13	0.20
	13	400	3000	0.16	0.23
	14	400	3750	0.23	0.30
	15	400	4500	0.38	0.38
Total Time(11-15)				1.02	1 28
Uncapacitated	16	400	1306	0.26	0.18
and lightly	17	400	2443	0.31	0 24
Capacitated	18	400	1306	0.36	0.14
problems	19	400	2443	0.57	0.23
(Fewer nodes)	20	400	1416	0.34	0.18
	21	400	2836	0.39	0.26
	22	400	1416	0.33	0.16
	23	400	2836	0.43	0.27
	24	400	1382	0.25	0.12
	25	400	2876	0.46	0.23
	26	400	1382	0.19	0.13
	27	400	2676	0.36	0.19
Total Time(16-27)				4.25	2.33
(Larger nodes)	28	1000	2900	1.00	0.33
	29	1000	3400	1.21	0.35
	30	1000	4400	1.72	0.44
	31	1000	4800	1.48	0.46
	32	1500	4342	1.76	0.52
	33	1500	4385	1.79	0.58
	34	1500	5107	1.80	0.69
	35	1500	5730	1.82	0.72
Total Time(28-35)				12.58	4.09

TABLE-3: This table contains Capacitated problems with large number of arcs with fewer nodes. These problems are generated with NETGEN.

All problems are generated using the Random number=13502460

Total-flow=400000 Cost-range=1-10000 MIN-CAP=16000 MAX-CAP=30000

NO OF SOURCES=8 NO OF SINKS =60

Percentage of arcs capacitated Min=30%, Max=20%

TABLE-4 This table contains Capacitated problems with large number of arcs with fewer number of nodes. Here we are having lesser flow values. These problems are generated with NETGEN using the Random number 13502460

Total-flow=10000 cost-range=1-10000 MIN-CAP=1600 MAX-CAP=3000

Percentage of arcs capacitated Min=30 %, Max=20 %

No of sources=8, No of Sinks =60

Table 5.4: Problems with lesser flow value

no	NODES	ARCS	Relax-code	Cycle-code	Optimal-value
1.	400	7000	0.64	0 81	29355904.00
2.	400	10000	0.92	1 17	24245933.00
3.	400	15000	1.31	1.72	10586044.00
4.	500	7000	0.90	1.00	56949065.00
5.	500	10000	1.11	1.24	20078692.00
6.	500	15000	1.38	1.87	18841929.00
7.	600	7000	0.94	1.32	47359349.00
8.	600	10000	1.00	1.75	47743815.00
9.	600	15000	1.41	1 87	22243930.00
10.	800	7000	0.60	1 91	60520678.00
11.	800	10000	1.20	2 00	54901837.00
12.	800	15000	1.54	2.42	26405504.00
Total time			12.95	19.08	

TABLE-5 This table contains transportation problems with larger number of arcs.

Total-flow=20000 cost-range=1-100

Random no for generating the problem is= 13502460

Table 5.5: Transportation problems

Problem no	No.of Sources	No.of Sinks	No.of Arcs	Relax-code	Cycle-code	Optimal-value
1	150	150	4000	0.48	0.61	250311.00
2	200	200	4000	0.50	0.72	312893.00
3	250	250	4000	0.61	1.10	375418.00
4	150	150	7000	0.52	0.75	179511.00
5	200	200	7000	0.63	0.81	210261.00
6	250	250	7000	0.77	1.23	253077.00
7	150	150	10000	0.82	1.11	128522.00
8	200	200	10000	0.91	1.32	166806.00
9	250	250	10000	1.00	1.56	200204.00
10	150	150	15000	1.12	1.71	119787.00
11	200	200	15000	1.23	2,31	133184.00
12	250	250	15000	1.32	2.51	135673.00
Total time				9.91	15.74	

In what follows we give the graphical representation of all the tables given above. It will give as a clear picture about the performance of the Relax-code and the Cycle-code to the corresponding problems.

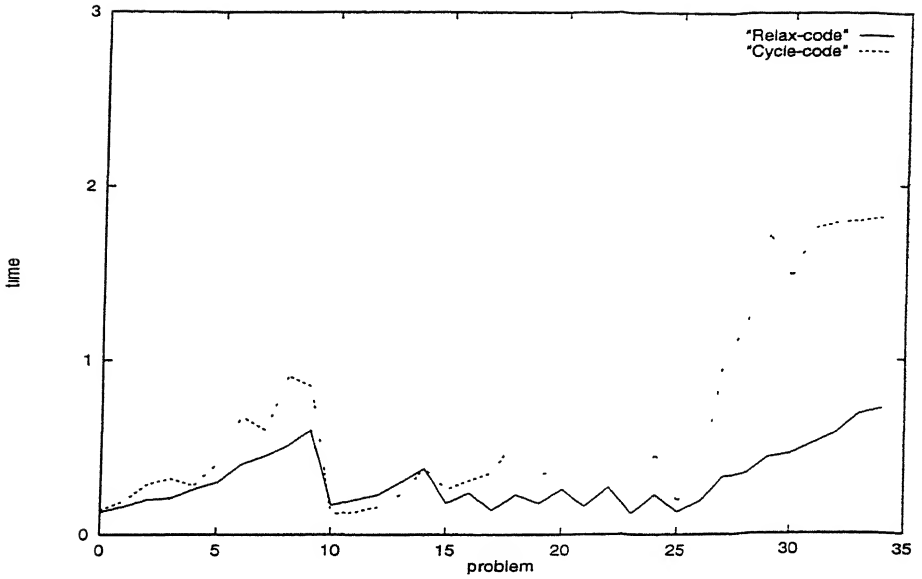


Figure 5.1: Bench Mark problems

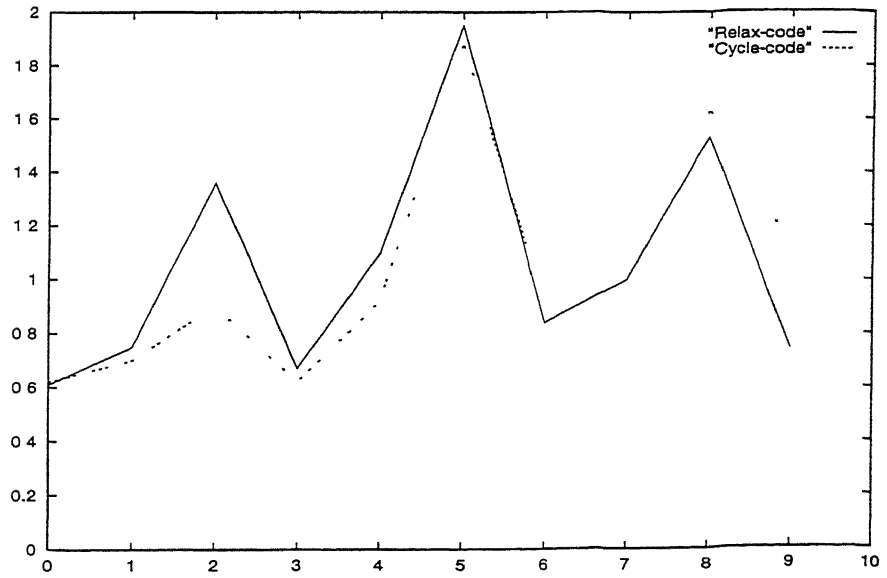


Figure 5.2: Assignment Problems

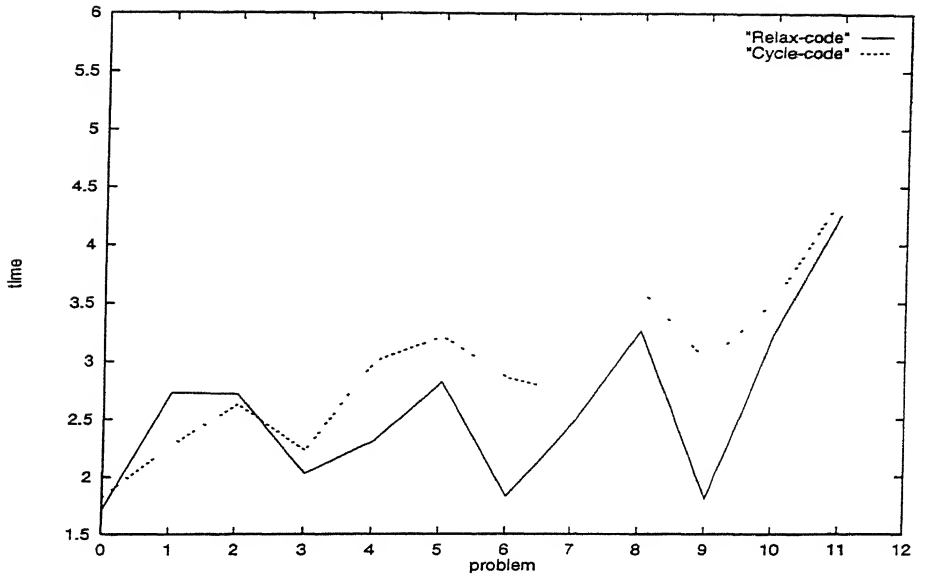


Figure 5.3: Problems with Larger number of arcs

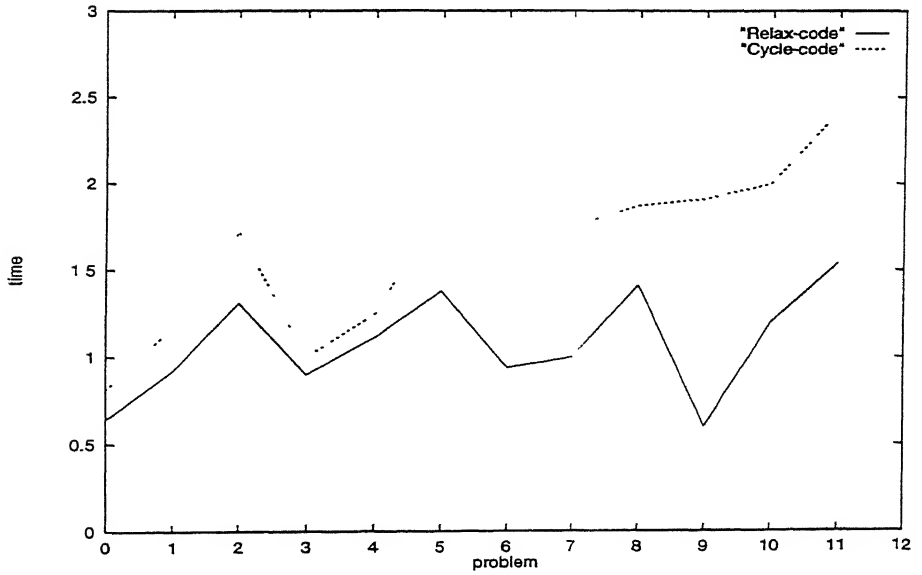


Figure 5.4: Problems with lesser flow values

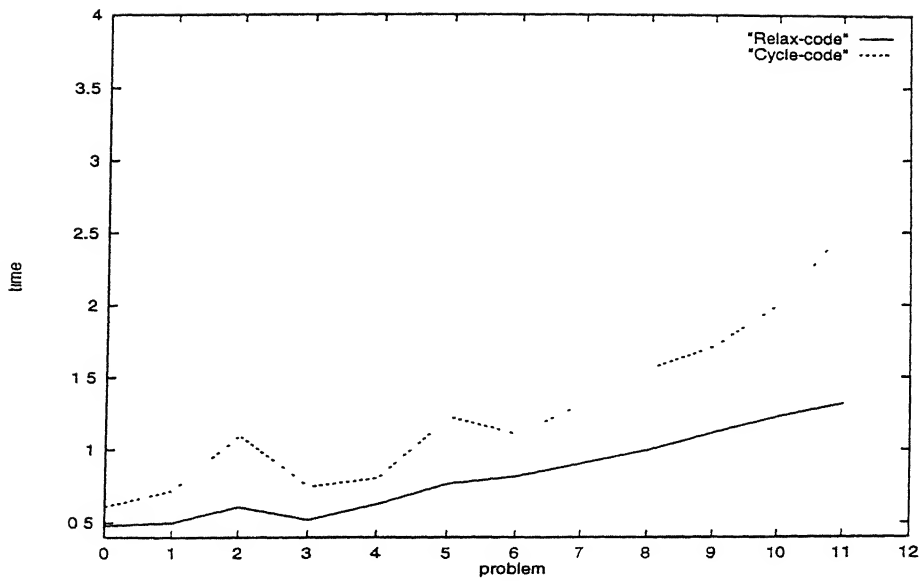


Figure 5.5: Transportation Problems

5.6 Submodular flows

Edmonds and Giles [18] introduced the submodular flow(SF) problem as a generalization of the network flow problem. In the SF, the netflows into the subsets are restricted by submodular set constraints. In order to formulate the Submodular flow problem we first give a few definitions.

Definition 5.6.1. *Let E denote a finite set, $\mathcal{F}$ a family of subsets of E.*
$f : \mathcal{F} \to \mathcal{R}$ is said to be submodular if for all $X, Y \in \mathcal{F}$,

$$f(X) + f(Y) \geq f(X \cup Y) + f(X \cap Y) \tag{5.6.1}$$

Two subsets $X, Y \in \mathcal{F}$ are said to be intersecting if none of X-Y, Y-X, $X \cap Y$ is empty.
$X, Y \in \mathcal{F}$ are said to be crossing if X, Y are intersecting and $X \cup Y \neq E$.
$f : \mathcal{F} \to \mathcal{R}$ is said to be intersecting submodular function if f satisfies (5.6.1) for all X, Y intersecting.
$f : \mathcal{F} \to \mathcal{R}$ is said to be crossing submodular function if f satisfies (5.6.1) for all X, Y crossing.

Consider a directed graph G(V,E), with node set V and arc set E, a submodular function b' defined on crossing family of subsets of V, lower and upper capacity vectors l and u, where $l : E \to (\mathcal{R}, -\infty)$ $u : E \to (\mathcal{R}, \infty)$ $l \leq u$, and a cost function $c : E \to \mathcal{R}$.
For a vector $x \in \mathcal{R}^E$ $\forall$ $A \subseteq V$, denote

$$\begin{aligned}
\rho_x(A) &= \Big(\sum x_{ij} : \ (i,j) \in E, \ (i,j) \text{ enters } A\Big) \\
\delta_x(A) &= \Big(\sum x_{ij} : \ (i,j) \in E, \ (i,j) \text{ leaves } A\Big) \\
\lambda_x(A) &= \rho_x(A) - \delta_x(A) \text{ and} \\
Q(l,u,b') &= \Big\{x \in \mathcal{R}^E : \ \lambda_x(A) \leq b'(A) \ \forall \ A \subseteq V \text{ and } l \leq x \leq u\Big\} \\
&\quad Q(l,u,b') \text{ is the submodular flow polyhedron.}
\end{aligned}$$

The SF problem is to

$$\text{minimize } cx$$

$$\text{subject to } x \in Q(l, u, b'). \tag{5.6.2}$$

A host of network flow problems such as the minimum cost flow problem (MCF) [20] , poly-matroidal network flow problem [41], the graph orientation problem [21], poly(matroid) intersection problem [19] ,the independent flow problem [24] and the directed cutting packing problem are all special cases of the SF problem.

5.6.1 Minimum cost-submodular flows

In this section we concentrate on the min-cost submodular flow problem, which is a special case of (5.6.2).

$$\text{Minimize} \quad z(x) = \sum_{(i,j)\in N} c_{ij}x_{ij} \tag{5.6.3}$$

subject to

$$\lambda_x(F) \leq b'(F) \;\forall\; F \subseteq V,\; F \neq \phi. V, b'(F) < \infty$$

$$\text{where} \quad 0 < x < u$$

A feasible solution of (5.6.3) is called a submodular flow.

A number of algorithms have been proposed for solving (5.6.3). Frank [23] and H. Rock et.el [29] have proposed strongly polynomial algorithms which rely on the ellipsoid algorithm. Other solution methodologies are negative circuit [58], primal simplex [6] , primal-dual [15] and out-of-kilter [26] which are all closely related to the corresponding algorithms for the Minimum cost network flow problems. These algorithms require an oracle which obtains the minimising set for a submodular function.

We have adapted the cycle cancelling algorithm of [52] for solving (5.6.3). Like all other algorithms for (5.6.3), this algorithm also requires an oracle for minimizing a submodular

function, but in terms of its implementation, it is simpler than the other algorithms. We make the following assumptions regarding (5.6.3).

1. All data (cost,capacity) are integer valued
2. the lower bound l_{ij} on arc flows are all zero.
3. the min-cost submodular polyhedron is nonempty and bounded.
4. A starting feasible flow x is available.

   In case a starting feasible flow is not available, the algorithm given in [22] can be used to obtain one.

Since it is simpler to deal with fully submodular functions rather than crossing or intersecting submodular functions, it has been shown in [27] that b' can be replaced by a unique fully submodular function b on the right hand side of (5.6.3), such that the set of feasible solutions remains the same, that is $Q(0, U, b') = Q(0\ U, b)$, and the optimal solution obtained with respect to Q(0,U,b) can be used for an optimal solution for (5.6.3) without any change. Obtaining a starting feasible solution for Q(0,U,b) is treated in [22]

5.6.2 Cycle cancelling algorithm

We will from now on refer to (5.6.3) with the right hand side function b' replaced by the fully submodular function b. For solving (5.6.3) we modify the cycle cancelling algorithm of [52] which is the basic algorithm for the algorithm dealt in the previous section for the minimum cost network flow problem. Our algorithm differs from that of Fujishige's [27] in its procedure for detecting negative cycles and is quite easy to implement. It should be possible to further improve our algorithm by adapting the Dynamic share procedure presented in the previous section, for the min-cost flow problem.

Definition 5.6.2. *For a directed graph G(V,E) a subset B of V is said to be a $u\bar{v}$ set if u is in B but not v.*

B a subset of V is said to be x-tight if $\lambda_x(B) = b(B)$.

It has been shown in [27] that union and intersection of x-tight sets are also x-tight.

For any feasible flow x, form the auxiliary graph $G' = (V, E')$ in the same manner as for the MCF in the previous section. with some additional arcs as follows:

1. For each arc (i,j) in E with flow $x_{ij} < u_{ij}$ introduce an arc (i,j) in G' with capacity $u_{ij} = u_{ij} - x_{ij}$ and cost c_{ij}.

2. For each arc (i,j)in E with flow $x_{ij} > 0$ introduce an arc (j,i) in G' with capacity $u_{ij} = x_{ij}$, with cost $-c_{ij}$.

3. For all $\{i, j\}$ in V ,if there is no $i\bar{j}$ tight set then introduce an arc (i.j) in G' with capacity $u_{ij} = min\,(b(F) - \lambda_x(F)|F \quad is \quad i\bar{j} \quad set)$ and cost $c_{ij} = 0$. This arc is called a jumping arc.

In the lemma given below we show that computing the capacity u_{ij} of a jumping arc requires minimizing a submodular function

Lemma 5.6.1. *For x a feasible flow for (5.6.3), and* $i,j \in V$

$$Minimising \quad (b(F) - \lambda_x(F)| \quad F \quad is \quad i\bar{j} \quad set) \tag{5.6.4}$$

is equivalent to minimizing a submodular function.

Proof. let $V' = V/(i, j)$

Define $f'(A) = (b - \lambda_x)(A \cup i)\ \forall\ A \subset V'$

f' is submodular on the subsets of V', because $(b - \lambda_x)$ is submodular on V, and

$\min\,(b(F) - \lambda_x(F)|F \quad is \quad (i\bar{j}) \quad set) = \min\{f'(A), A \subseteq V'\}$

Hence the lemma. □

5.6.3 Optimality conditions

Let B be a (0 ± 1) matrix with rows corresponding to sets X. with $b(X) < \infty$. The dual of problem (5.6.3) is

$$\begin{aligned} &Maximize \quad -by - \alpha u \\ &\text{subject to} \quad -yB - \alpha I \leq c \quad y, \alpha \geq 0 \end{aligned} \tag{5.6.5}$$

where the variable α corresponds to the arcs in E and y to the subsets of V. I denotes the identity matrix of appropriate size.

Let us introduce the potential vector $\pi \in \mathcal{R}^V$, to the nodes of V and define $yF(e) = \pi(j) - \pi(i)$, for all $e = (i, j) \in E$ For details see [18].

The complementary slackness conditions, for the primal dual pair (5.6.3),(5.6.5) as in [15] are·

1. $c_{ij} - \pi(i) + \pi(j) < 0 \Rightarrow x_{ij} = u_{ij}$.
2. $c_{ij} - \pi(i) + \pi(j) > 0 \Rightarrow x_{ij} = 0$.
3. $yF > 0 \Rightarrow \lambda_x(F) = b(F)$,
   that is, $\forall i, j \in V \quad if\ \pi(j) > \pi(i) \quad$ then
   $min(b(F) - \lambda_x(F) | F \quad \text{is} \quad i\bar{j} \quad \text{set}) = 0$.

thus in order to solve (5.6.3) we need to find a x feasible for (5 6.3) and π a set of node potentials satisfying the complementary slackness conditions (1),(2) and (3). We now define the reduced cost optimality conditions and the ϵ-optimality conditions, and it is the ϵ - optimality conditions that we will be working with.

Reduced cost optimality conditions.

As for as the MCF problem, by defining $c^{\pi}_{ij} = c_{ij} - \pi_i + \pi_j$. the complementary slackness conditions (1),(2) and (3) for the pair x,π can be rewritten as

1. $c_{ij}^{\pi} \geq 0 \ \forall \ arc \ (i,j) \ in \ G'(x)$

2. $\forall \ i,j \ \in N, \ \pi(j) > \pi(i)$, then
   $min(b(F) - \lambda_x(F) \mid F \quad i\bar{j} \quad \text{set}) = 0$

Details for combining complementary slackness conditions (1) and (2) into the reduced cost optimality condition (1) can be found in [3].

ϵ -optimality.

A flow x is said to be ϵ-optimal for some $\epsilon > 0$, if together with some potential vector π, it satisfies the following ϵ-optimality conditions.

1. $c_{ij}^{\pi} \geq -\epsilon \ \forall \ arc(i,j) \ in \ G(x)$

2. $\forall i,j \in V$, such that $\pi(j) > \pi(i)$,
   $min(b(F) - \lambda_x(F) | F \quad (ij') \quad \text{set}) = 0$

We now show that with integer costs, an ϵ optimal flow x is optimal if $\epsilon < 1/n$.

Theorem 5.6.1. *For a minimum cost submodular flow problem with integer costs. any ϵ optimal feasible flow is optimal if $\epsilon < 1/n$.*

Proof. For any directed cycle W in $G'(x)$, we have $\sum_{(i,j)\in W} c_{ij}^{\pi} \geq -n\epsilon > -1$. Since the costs are integers this implies that there are no negative cycles in $G'(x)$. For details see (Lemma 10.2) of [3]. So reduced cost optimality condition (1) is satisfied. ϵ-optimality condition (2) is the same as the reduced cost optimality condition (2). Hence any ϵ-optimal flow with $\epsilon < 1/n$ is optimal for (5.6.3) Hence the theorem. □

Definition 5.6.3. *An arc (i,j) in the auxillary graph $G'(x)$ is called an admissible arc if $c_{ij}^{\pi} < 0$.*

We call a graph with only admissible arcs as admissible graph.

We call the cycles in the admissible graph(consisting of admissible arc) as admissible cycles.

Cycle cancelling algorithm by sharing optimality violation:

Let x be a feasible flow obtained by using the algorithm of [22]. Construct the auxillary graph $G'(x) = (V, E')$.

Initialy, take the potential $\pi_i = 0$ for all nodes $i \in V$

The reduced cost for (i,j) $\in E'$ are,

$c_{ij}^{\pi} = c_{ij} - \pi_i + \pi_j$

Let us make the admissible graph acyclic by augmenting the flow along the cycles, present in the admissible graph.

We then choose a most negative eligible arc say (q,p) and find the set of nodes S reachable from p through eligible arcs only.

Let $\bar{S} = V - S$.

Define ϵ as follows.

$\epsilon_1 = -c_{qp}^{\pi}$

$\epsilon_2 = min\{\pi(j) - \pi(i) \mid \text{for } i \in S, j \in \bar{S} \text{ with}$

$\pi(j) > \pi(i),$ and $min\,(b(F) - \lambda_x(F) \mid F \text{ is } \bar{ij} \text{ set}) = 0\}$
(this will ensure that the ϵ – optimality condition (2) will continue to be satisfied)

$\epsilon = min(\epsilon_1\ ,\ \epsilon_2)$

Now for every node i in S update the potential as follows.

$\pi(i) = \pi(i) + \frac{\epsilon}{2}$.

Note that $\pi_j > \pi_i$ after the update.

The update will make

(i) the reduced cost of every arc from S to $\bar{S}$ reduce by $\frac{\epsilon}{2}$ and

(ii) the reduced cost of every arc from $\bar{S}$ to S increase by $\frac{\epsilon}{2}$.

Updating the potential may create new eligible arcs in $(S, \bar{S})$, while keeping atleast the arc (q,p) eligible. We try to create new eligible arcs by this process, and due to this the admissible graph becomes cyclic. We make it again acyclic by repeatedly finding and

cancelling admissible cycles.

We cancel the admissible cycles by augmenting the flow as follows:

Let

$$\delta_1 = \min\{u_{ij} - x_{ij} \mid (i,j) \text{ is of type (1) in the auxillary graph}\}$$

$$\delta_2 = \min\{x_{ij} \mid (i,j) \text{ is of type (2) in the auxillary graph}\}$$

$$\delta_3 = \min\{min(b(F) - \lambda_x(F) \mid F \text{ is } ij' \text{ set}) \text{ for all jumping arcs}\}$$

$$\text{and } \delta = \min\{\delta_1,\ \delta_2,\ \delta_3\}$$

For finding δ_3 we would need an oracle for minimizing a submodular function.

The modified flow is

$$x'_{ij} = \begin{cases} x_{ij} + \delta, & (i,j) \text{ is of type (1) in the auxillary graph} \\ x_{ij} - \delta, & (i,j) \text{ is of type (2) in the auxillary graph} \\ x_{ij}, & (i,j) \text{ is a jumping arc} \end{cases}$$

While augmenting flow from i to j, one may come across parallel arcs and the natural choice would be to choose the one with maximum capacity. Henceforth, while augmenting flow along a cycle we will choose the arc with maximum capacity.

If the admissible graph is acyclic find the node p such that the most negative arc is an incoming arc to node p, and repeat the above process of updating the potentials.creating new eligible arcs, augmenting the flow and finding new auxillary graph until $\epsilon \geq 1/n$. Stop the process when $\epsilon < 1/n$.

We now prove the validity of our algorithm.

Theorem 5.6.2. *The flow x and the potential vector π obtained at the end of our algorithm satisfy the reduced optimality conditions.*

Proof. It is clear from the algorithm that at each stage the pair (x, π) maintains ϵ-optimality for some $\epsilon > 0$. The algorithm terminates only when ϵ reduces to a value less than 1/n.Thus it follows from theorem 5.6.1 that our algorithm terminates at an optimal solution x for (5.6.3). □

We now give a concise version of our algorithm .

Algorithm

begin

$\pi = 0$, x be any feasible flow

Cancel admissible cycles by augmenting flow until
the admissible graph become acyclic.

while $(\epsilon \geq 1/n)$ do

begin

(i) Create admissible arcs as explained above, and update the potentials.

(ii) Cancel admissible cycles.

(iii) Form auxillary graph with respect to the new flow x

end;

end;

One can note that after n-consecutive iterations of the algorithm, the reduced cost of the most negative arc in G(x) increases, and hence the value of ϵ will decrease. The integrality of costs will ensure the finiteness of the algorithm.

5.6.4 Complexity

Assume that an oracle for minimizing a submodular function is available to us with time complexity o(k).

Since the starting value of ϵ is max $\{c_{ij}\}$=C, and after atmost n consecutive iterations, the value of the most negative arc reduces to half of its value, after n(log(nC)) iteration the value will be $\geq -1/n > -\epsilon$.

The number of augmentation is therefore atmost o(nm), and each augmentation takes o(nh) time.Thus the upper bound on the flow augmentation is $o(n^2mk)$.

5.7 Concluding Remarks

From the results based on Chapter-2 and Chapter-3. it is very clear that for the submodular and bisubmodular functions the values of f at the empty set is playing a vital role. Again the submodular function f having non zero value at the empty set can be handled easily compared to a bisubmodular function f having non zero value at the empty set. For the bisubmodular function f, if $f(\phi, \phi) < 0$ solving the LPP over the associated polyhedron seems to be a difficult problem. We feel that in this case also it should be possible to exploit the polyhedral structure of the constraint set so that one can develop an efficient algorithm for the corresponding LPP.

Our algorithm for solving an LPP over a submodular polyhedron relies heavily on an oracle for minimising a submodular function.

A polynomial time algorithm for minimising a submodular function will speed up our algorithm and many other algorithms for which it is need as an oracle.

From the discussions on increasing bisubmodular function in chapter4 we observe that the polyhedral structure of an increasing bisubmodular polyhedron is similar to that of a submodular polyhedron in many ways.

Minimising a submodular function can be reduced to a membership problem for a polymatroid polyhedron. The question to be asked is that can minimisation of a bisubmodular function be also reduced to a membership problem over an increasing bisubmodular polyhedron.

The cycle cancelling algorithm in chapter-5 for the minimum cost network flow problem simultaneously finds negative cycles and tightens optimality violations by using the basic graph search algorithm. The algorithm is very simple to implement and from the computational results we obtain, behaviour of our algorithm is very efficient. We also extend this algorithm to solve the minimum cost submodular flow problem.

It should be possible to further improve our algorithm for solving the minimum cost submodular flow problem by adapting the Dynamic share procedure presented in this chapter, for the minimum cost network flow problem.

Bibliography

[1] H. A. Aashitiani and T. L. Magnanti, *Implementing primal-dual network flow algorithm*, Technical Report OR 055-76 operations research center MIT, Cambridge, MA, (1976).

[2] R. K. Ahuja, M. Kodialam, A. K. Mishra, and J. B. Orlin. *Computational investigations of maximum flow algorithm.* PrePrint. (1996).

[3] R. K. Ahuja. T. L. Magnanti, and J. B. Orlin, *Network flows: Theory, Algorithms and Applications*, Prentice Hall, Englewood Cliffs, NJ. 1993.

[4] K. Ando and S. Fujishige, *On structures of bisubmodular polyhedra.* Mathematical Programming, 74 (1996), pp. 293–317.

[5] K. Ando and S. Fujishige, *On structures of bisubmodular polyhedra*, Report No-94821, Institute för Oknometric UND Operations Research, Universitat Bonn, (April 1994).

[6] F. Barahona and W. H. Cunningham, *A submodular network simplex method*, Mathematical programming study, 22 (1984), pp. 9–31.

[7] F. Barahona and E. Tardos, *Note on weintraub's minimum cost circulation algorithm*, SIAM journal of Computing, 18 (1989), pp 579–583.

[8] D. P. Berstsekas and P. Tseng, *Relaxation methods for minimum cost ordinary and generalised network flow problems*. Operations Research journal, 36 (1988), pp. 93–114.

[9] R. E. Bixby, W. H. Cunningham, and D. M. Topkis, *Partial order of a polymatroid extreme point*, Mathematics of Operations Research, 10 (1985), pp. 367–378.

[10] R. G. Bland and M. L. Vergnas, *Orientability of matroids*, Journal of Combinatorial theory, B 24 (1978), pp. 94–123.

[11] A. Bouchet and W. H. Cunningham, *Delta matroids, jump systems and bisubmodular polyhedra*, Preprint, (1994).

[12] R. chandrasekaran and S. N. Kabadi. *Psedomatroids*, Discrete mathematics, 71 (1988), pp. 205–217.

[13] W. H. Cunningham, *Testing membership in matroid polyhedron*, Journal of Combinatorial Theory, B36 (1984), pp. 161–188.

[14] W. H. Cunningham, *On submodular function minimisation*, Combinatorica, 5 (1985), pp. 185–192.

[15] W. H. Cunningham and A. Frank, *A primal-dual algorithm for submodular flows.*, Mathematics of Operations Research, 10 (1985), pp 251–262.

[16] W. H. Cunningham and J. G. Krotki. *b-matching degree sequence polyhedra*, Combinatorica, 11 (1991), pp. 219–230.

[17] F. D. J. Dunstan and D. J. A. Welsh, *A greedy algorithm solving a certain class of linear programmes*, Mathematical Programming, 5 (1973), pp. 338–353.

[18] EDMONDS AND GILES, *A min-max relation for submodular functions on graphs.*, Annals of Discrete Mathematics, 1 (1977), pp. 185–204.

[19] J. EDMONDS, *Submodular function, matroids and certain polyhedra*, in Proceedings of the Calgary international conference on Combinatorial structures and their applications, Newyork., 1970, pp. 69–87.

[20] L. R. FORD AND D. R. FULKUERSON, *Flows in networks*, Princeton university press, Princeton, N.J, (1962).

[21] A. FRANK, *A note on k-strongly connected orientations of an undirected graph*, Discrete Mathematics, 39 (1982), pp. 103–104.

[22] A. FRANK, *Finding feasible vectors of Edmonds-Giles polyhedra*, Jl of combinatorial theory. B36 (1984), pp. 221–239.

[23] A. FRANK AND E. TARDOS, *An application of the simultaneous approximation in combinatorial optimization*, Report No. 85375. Institute *für ö* Knometric and Operations Research, university Bonn. (1985).

[24] S. FUJISHIGE, *The independent-flow problems and submodular functions(in japanese)*, Journal of the faculty of Engineering, university of Tokyo, A-16 (1978), pp. 42–43.

[25] S. FUJISHIGE, *A characterization of faces of the base polyhedron associated with a submodular system*, Journal of the O.R society of Japan, 27 (1984), pp. 112–128.

[26] S. FUJISHIGE, *An out-of-kilter method for submodular flows*, Discrete Applied Mathematics, 17 (1987), pp. 3–16.

[27] S. FUJISHIGE, *Submodular functions and optimization*, Elsevier science publishers B.V, 655, Avenue of the Americes Newyork. N.Y.10010, USA, 1990.

[28] S. FUJISHIGE, *A min-max theorem for bisubmodular polyhedra*, Research Institute for Discrete Mathematics, Institute for Operations Research, University of Bonn, Report No 94818, (1994).

[29] S. FUJISHIGE, H. ROCK, AND U. ZIMMERMANN, *A strongly polynomial algorithm for minimum cost submodular flow problem*, Mathematics of Operations Research, 14 (1989), pp. 60–69.

[30] A. V. GOLDBERG AND R. E. TARJAN, *Finding minimum cost circulation by cancelling negative cycles*, Journal of ACM, 36 (1989), pp. 873–886.

[31] A. V. GOLDBERG AND R. E. TARJAN, *Solving minimum cost flow problem by successive approximation*, Mathematics of Operations Research, 15 (1990), pp. 430–466.

[32] M. GRÖTSCHEL, L. LOVASZ, AND A. SCHRIJVER. *Geometric Algorithms and Combinatorial Optimization (Algorithms and combinatorics 2)*, Springer, Berlin, 1988.

[33] R. GUHA, S. N. KABADI, AND P. SHARMA, *Bisubmodular polyhedra: facets, adjacency and a greedy algorithm*, In XVth internat. Symp. on Mathematical Programming, Ann Arbor, MI, 1994.

[34] R. K. GUHA, *On Ditroids and Bisubmodular systems*, PhD thesis, IIT-Kanpur, 1994.

[35] M. IRI, *Applications of matroid theory*, Mathematical Programming-The state of art(A.Bachem, M.grotschel and B.Korte, eds., Springer, Berlin,), (1983), pp. 158–201.

[36] S. N. KABADI, *Characterization and development of solution methods for special classes of integral systems*, Dissertation, School Management, University of Texas at Dallas, (1984).

[37] S. N. KABADI AND R. CHANDRASEKAREN, *On totally dual integral systems*, Discrete Applied Mathematics, 26 (1990), pp. 87–104.

[38] R. M. KARP AND J. B. ORLIN, *Parametric shortest path algorithms with an application to cyclic staffing*, Discrete Applied Mathematics, 3 (1981), pp. 37–45.

[39] M. KLEIN, *A primal method for minimal cost flow with application to the assignment and transportation problems*, Management Science, 14 (1967), pp. 205–220.

[40] D. KLINGMAN, A. NAPIER, AND J. STUTZ, *Netgen: A program for generating large scale (un)capacitated assignment transportation and minimum cost flow network problems.*, Management science, 20 (1974), pp. 814–822.

[41] E. L. LAWLER AND C. U. MARTEL, *Computing maximal 'polymatroidal' network flows*, Mathematics of operations Research, 7 (1982), pp. 334–347.

[42] L. LOVASZ, *Submodular functions and convexity*, Mathematical Programming-The state of the Art(A. Bachem, M. Grotschel and B. Korte, eds.. Springer, Berlin,), (1983), pp. 235–257.

[43] K. MUROTA, *Systems analysis by graphs and matroids*, Algorithms and Combinatorics 3 (Springer, Berlin,), (1987).

[44] H. NARAYANAN, *The principle lattice of partition of a submodular function*, Linear Algebra and its Applications, 144 (1991), pp 179–216.

[45] G. L. NEMHAUSER AND L. A. WOLSEY, *Integer and Combinatorial Optimization*, John wiley and Sons, Newyork, 1988.

[46] L. QI, *Directed submodularity, ditroids and directed submodular flows*, Mathematical Programming, 42 (1988), pp. 579–599

[47] L. QI, *Odd submodular functions, Dilworth-function and discrete convex function*, Mathematics of Operations Research, 13 (1988), pp. 435–446.

[48] A. RECSKI, *Matroid theory and applications*, Springer.

[49] A. SCHRIJVER, *Proving total dual integrality with cross free families, a general frame work*, Mathematical Programming, 29 (1984). pp 15–27.

[50] A. SCHRIJVER, *Total dual integrality from directed graphs, crossing families and sub and supermodular functions,*, in W.R. Pullyblank ed Progress in Combinatorial Optimization, (Acadamic Press, Toronto, ontario) (1984), pp. 315–362.

[51] M. SOHONI, *Membership in submodular and other polyhedra*, Tech. Rep. TR-102-92, IIT, Bombay, 1994.

[52] P. T. SOKKALINGAM, *The minimum cost flow problem: Primal Algorithms and Cost Perturbation*, PhD thesis, IIT-Kanpur, 1995.

[53] E. TARDOS, *Generalised matroids and submodular colorings*, Tech. Rep. AE 19/83, University of Amesterdam.

[54] D. M. TOPKIS, *Adjacency on polymatroids*, Mathematical Programming, 30 (1984), pp. 229–237.

[55] C. WALLACHER AND U. T. ZIMMERMANN, *A combinatorial interior point method for network flow problems*, in Presented at the 14th International Symposium on Math. Programming Amsterdam, The Netherlands, 1991.

[56] D. J. A. WELSH, *Matroid theory*, Acadamic Piess, London, (1976).

[57] S. ZHANG, *On vertices adjacency characterization and optimization of polymatroid*, in Report No. 8916/A, Erasmus University, Rotterdam

[58] U. ZIMMERMANN, *Minimisation on submodular flows*. Discrete Applied Mathematics, 4 (1982), pp. 303–323.

APPENDIX

Code for our cycle cancelling algorithm given in chapter5

```
#include "stdio.h"
#include "time.h"
#include <unistd.h>
#define  infty  500000
#define true 1
#define false 0
void  max_flow(void);
void max_augument(void);
int max_exam(int i);
void min_cost(void);
int max_viol(void);
int p2greater(unsigned int i);
int find(void);
void sscan(int p);
int exam(int i);
void ini_data(void);
void augument(int i1,int i2, int j1);
void shortest_path(int s);
void augument_path(int i1,int i2);
int tail[51000],head[51000],cost[51000],cap[51000],
int trace[51000],rpoint[4000],crpoint[4000];
int point[4000],flow[51000],cpoint[4000];
int a[4000][250];
int cou[4000],distance[4000],perm[4000];
float mctime;
int potent[51000],b[4000];
int scal_fact;
int le_arc ,total_cost,current;
int nodes,arcs,nsorc,nsink;
int pred[4000],link[51000],label[4000];
int cval,outflow,key,scounter,current_node;
void main(void)
{
        int i,j,k,rcost,counter;
        FILE *fp;
        fp=fopen("data", "r");
        fscanf(fp,"%d\t %d\t %d\t %d\n", &nodes,&arcs,&nsorc,&nsink);
        for( i=1 ;i< arcs+1; ++i)
        {
           fscanf(fp,"%d\t %d\t %d\t %d\n ", &tail [i],&head[i],&cost[i],&cap[i.
        }
        fclose(fp);
        ini_data();
        max_flow();
        min_cost();
        total_cost=0;
        for(j=1;j<arcs+1; ++j)
        {
                if (flow[j]>0)
                        total_cost=total_cost+(flow[j]*cost[j]);
        }
        printf("total_cost=%d\n ", total_cost);
```

```
}
-----------------------------------------------
This subroutine is to initialize the datas
-----------------------------------------------
void ini_data(void)
{
        int i,j,k;
        for (i=1;i<nodes+4;++i)
                cou[i]=1;
        for (j=1; j<arcs+1; ++j)
        {
                i=head[j];
                a[i][cou[i]]=j;
                cou[i]=cou[i]+1;
        }
        i=1;
        j=1;
        k=1;
        do
        {
                while(a[i][j]!=0)
                {
                        trace[k]=a[i][j];
                        j=j+1;
                        k=k+1;
                }
                i=i+1;
                j=1;
        }while(a[i][j]!=0);
        i=nodes+2;
        j=1;
        while(a[i][j]!=0)
        {
                trace[k]=a[i][j];
                j=j+1;
                k=k+1;
        }
        point[1]=1;
        rpoint[1]=1;
        k=2;
        j=2;
        for( i=1;  <arcs+1; ++i)
        {
                if (head[trace[i]]!=head[trace[i+1]])
                {
                        rpoint[j]=i+1;
                        j=j+1;
                }
                if (tail[i]!=tail[i+1])
                {
                        point[k]=i+1;
                        k=k+1;
                }
        }
        rpoint[nodes+2]=rpoint[nodes+1];
        point[nodes+2]=arcs+1;
        rpoint[nodes+3]=arcs+1;
```

```
}
-----------------------------------------------------
This Subroutine will calculate the starting feasible
flow by solving the max-flow problem using labelling
Algorithm.
-----------------------------------------------------

void max_flow(void)
{
        int i,j,k,delta,t;
        for (i=1; i<(nodes+4);++i)
        {
                cpoint[i]=point[i];
                crpoint[i]=rpoint[i];
                pred[i]=0;
        }
        for (j=1;j<arcs+1; ++j)
        {
                flow[j]=0;
                link[j]=0;
        }
        pred[nodes+1]=-1;
        delta=0;
        j=nodes+1;
        do
        {
                i=j;
                j=max_exam(i);
                if (j==nodes+2)
                {
                        max_augument();
                        j=nodes+1;
                }
        } while(j!=-1);
}
int max_exam(int i)
{
        int carc;
        int j,links,k;
        while(cpoint[i]<point[i+1])
        {
                carc=cpoint[i];
                j=head[carc];
                if ((flow[carc]<cap[carc])&&(pred[j]==0))
                {
                        pred[j]=i;
                        link[j]=carc;
                        return(j);
                }
                else
                {
                        cpoint[i]=cpoint[i]+1;
                }
        }
        while(crpoint[i]<rpoint[i+1])
        {
```

```
                carc=trace[crpoint[i]];
                k=tail[carc];
                if ((flow[carc]>0)&&(pred[k]==0))
                {
                        pred[k]=i;
                        link[k]=carc;
                        return(k);
                }
                else crpoint[i]=crpoint[i]+1;
        }
        j=pred[i];
        pred[i]=0;
        links=link[i];
        if (tail[link[i]]==j)
                ++cpoint[j];
        else ++crpoint[j];
        return j;
}
void max_augument(void)
{
        int delta,temp;
        int p,k,i;
        delta=infty;
        p=nodes+2;
        while(p!=nodes+1)
        {
                k=link[p];
                if (head[k]==p)
                {
                        if (delta>cap[k]-flow[k]) delta=cap[k]-flow[k];
                }
                else
                {
                        if (delta>flow[k])  delta =flow[k];
                }
                p=pred[p];
        }
        p=nodes+2;
        while(p!=nodes+1)
        {
                k=link[p];
                if (head[k]==p)
                {
                        flow[k]+=delta;
                }
                else
                {
                        flow[k]-=delta;
                }
                cpoint[p]=point[p];
                crpoint[p]=rpoint[p];
                temp=pred[p];
                pred[p]=0;
                p=temp;
        }
```

```
        for (p=1;p<nodes+3;++p)
        {
                cpoint[p]=point[p];
                crpoint[p]=rpoint[p];
        }
        pred[nodes+1]=-1;
}
-------------------------------------------------------
Subroutine for  the minimum-cost flow problem using
our new cycle cancelling algorithm disscused in
chapter5
-------------------------------------------------------

void min_cost(void)
{
        int d,j,i,max,k,rcost,s;
        max=max_viol();
        scal_fact=p2greater(max);
        scal_fact=scal_fact/2;
        le_arc=0;
        while(scal_fact>=1)
        {
                do
                {
                        i=find();
                        if (i != nodes+4)
                        {
                                pred[i]=-1;
                                sscan(i);
                        }
                }while(i !=nodes+4);
                le_arc=0;
                scal_fact=scal_fact/2;
        }
        for(i=1;i<nodes+3;++i)
        {
                b[i]=0;
        }
        for(j=1;j<arcs+1;++j)
        {
                rcost=cost[j]-potent[tail[j]]+potent[head[j]];
                if((flow[j]<cap[j])&&(rcost==-1))
                {
                        b[tail[j]]=b[tail[j]]-(cap[j]-flow[j]);
                        b[head[j]]=b[head[j]]+(cap[j]-flow[j]);
                        flow[j]=cap[j];
                }
                if((flow[j]>0)&&(rcost==1))
                {
                        b[tail[j]]=b[tail[j]]+flow[j];
                        b[head[j]]=b[head[j]]-flow[j];
                        flow[j]=0;
                }
        }
        current_node=1;
        do
        {
```

```
                if(b[current_node]>0)
                {
                        shortest_path(current_node);
                        if(b[current_node]>0)
                                current_node=current_node-1;
                }
                ++current_node;
        } while(current_node!=nodes+3);
}
void shortest_path(int s)
{
        int i,j,dc,smalldis;
        int carc;
        int newdis,k;
        for(i=0;i<nodes+3;++i)
        {
                distance[i]=infty;
                perm[i]=0;
                pred[i]=0;
                link[i]=0;
                label[i]=0;
                cpoint[i]=point[i];
                crpoint[i]=rpoint[i];
        }
        scounter=2;
        label[1]=s;
        perm[s]=1;
        distance[s]=0;
        current=s;
        do
        {
                smalldis=infty;
                dc=distance[current];
                while(cpoint[current] < point[current+1])
                {
                        carc=cpoint[current];
                        j=head[carc];
                        if((perm[j]==0)&&(flow[carc]<cap[carc]))
                        {
                                if(pred[j]==0)
                                {
                                        label[scounter]=j;
                                        ++scounter;
                                }
                                newdis=dc+(cost[carc]-potent[current]+potent[j]
                                if (newdis<distance[j])
                                {
                                        distance[j]=newdis;
                                        Pred[j]=current;
                                        link[j]=carc;
                                }
                        }
                        ++cpoint[current];
                }
                while(crpoint[current] < rpoint[current+1])
```

```
                {
                        carc=trace[crpoint[current]];
                        j=tail[carc];
                        if((perm[j]==0)&&(flow[carc]>0))
                        {
                                if(pred[j]==0)
                                {
                                        label[scounter]=j;
                                        ++scounter;
                                }
                                newdis=dc-(cost[carc]-potent[j]+potent[curren
                                if(newdis<distance[j])
                                {
                                        distance[j]=newdis;
                                        pred[j]=current;
                                        link[j]=carc;
                                }
                        }
                        ++crpoint[current];
                }
                for(i=1;i<scounter+1; ++i)
                {
                        if(perm[label[i]]==0)
                        {
                                if(smalldis>distance[label[i]])
                                {
                                        smalldis=distance[label[i]];
                                        current=label[i];
                                }
                        }
                }
                perm[current]=1;
        }while(b[current]>=0);
        augument_path(current,s);
}
void augument_path(int i1,int i2)
{
        int i,j,p,delta;
        delta=infty;
        p=i1;
        do
        {
                j=link[p];
                if(head[j]==p)
                {
                        if(delta>(cap[j]-flow[j]))
                                delta=cap[j]-flow[j];
                }
                else if(tail[j]==p)
                {
                        if (delta>flow[j]) delta=flow[j];
                }
                p=pred[p];
        } while(p!=i2);
        if(delta>b[i2]) delta=b[i2];
```

```
        if(delta>-b[i1]) delta=-b[i1];
        p=i1;
        do
        {
                j=link[p];
                if(head[j]==p)
                        flow[j]=flow[j]+delta;
                else flow[j]=flow[j]-delta;
                p=pred[p];
        } while(p!=i2);
        b[i2]=b[i2]-delta;
        b[i1]=b[i1]+delta;
        for(i=1;i<scounter+1; ++i)
        {
                j=label[i];
                if(perm[j]!=0)
                {
                        potent[j]=potent[j]-distance[j]+distance[i1];
                }
        }
}
int max_viol(void)
{
        int j;
        int max;
        max=0;
        for(j=1; j<arcs+1;j++)
        {
                if((flow[j]<cap[j])&&(cost[j]<0)&&(max<-cost[j]))
                        max=-cost[j];
                else
                {
                        if((flow[j]>0)&&(cost[j]>0)&&(max<cost[j]))
                                max=+cost[j];
                }
        }
        return max;

int p2greater( unsigned int i)

        int k,k1;
        k=0;
        k1=0;
        while(i)
        {
                i=i>>1;
                ++k;
        }
        k1=1<<k;
        return k1;

find(void)

        int j,rcost;
        j=le_arc;
```

```
        while(j<arcs+1)
        {
                j=j+1;
                rcost=cost[j]-potent[tail[j]]+potent[head[j]];
                if ((flow[j]<cap[j])&&( rcost< -scal_fact))
                {
                        le_arc=j;
                        return head[j];
                }
                else if ((flow[j]>0)&& (rcost>scal_fact))
                {
                        le_arc=j;
                        return tail[j];
                }
        }
        return nodes+4;
}
void sscan(int p)
{
        int j;
        j=p;
        do
        {
                j=exam(j);
        } while(j != -1);
}
int exam(int i)
{
        int k,j,carc;
        while(cpoint[i]<point[i+1])
        {
                carc=cpoint[i];
                j=head[carc];
                if( flow[carc]<cap[carc])
                {
                        if((cost[carc]-potent[i]+potent[j])< 0)
                        {
                                if( pred[j] !=0)
                                {
                                        augument(i,j,carc);
                                        return j;
                                }
                                else
                                {
                                        pred[j]=i;
                                        link[j]=carc;
                                        return j;
                                }
                        }
                }
                cpoint[i]=cpoint[i]+1;
        }
        while (crpoint[i]<rpoint[i+1])
        {
                carc=trace[crpoint[i]];
```

```
                if(flow[carc]>0)
                {
                        k=tail[carc];
                        if((cost[carc]-potent[k]+potent[i]) > 0)
                        {
                                if(pred[k]!= 0)
                                {
                                        augument(i,k,carc);
                                        return k;
                                }
                                else
                                {
                                        pred[k]=i;
                                        link[k]=carc;
                                        return k;
                                }
                        }
                }
                crpoint[i]=crpoint[i]+1;
        }
        potent[i]=potent[i]+scal_fact;
        cpoint[i]=point[i];
        crpoint[i]=rpoint[i];
        j=pred[i];
        pred[i]=0;
        return j;
}
void augument(int i1, int i2, int j1)
{
        int p,delta,j,temp;
        p=i1;
        delta=infty;
        do
        {
                j=link[p];
                if(head[j]==p)
                {
                        if (delta>(cap[j]-flow[j]))
                                delta=cap[j]-flow[j];
                }
                else if (tail[j]==p)
                {
                        if (delta>flow[j]) delta=flow[j];
                }
                p=pred[p];
        }while (p !=i2);
        if(tail[j1]==i1)
        {
                if((cap[j1]-flow[j1])<delta) delta=cap[j1]-flow[j1];
                flow[j1]+=delta;
        }
        else
        {
                if (flow[j1]<delta)
                        delta=flow[j1];
```

```
                flow[j1]-=delta;
        }
        p=i1;
        do
        {
                j=link[p];
                if (head[j]==p) flow[j]+=delta;
                else flow[j]-=delta;
                temp=p;
                p=pred[p];
                pred[temp]=0;
        }while(p != i2);
}
```